UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

JOELSON PERDONÁ SERAFIN

MANIPULAÇÃO DE AUTÔMATOS FINITOS NO AFLAB

CRICÍUMA, JULHO DE 2009

JOELSON PERDONÁ SERAFIN

MANIPULAÇÃO DE AUTÔMATOS FINITOS NO AFLAB

Trabalho de Conclusão de Curso apresentado para obtenção do Grau de Bacharel em Ciência da Computação da Universidade do Extremo Sul Catarinense.

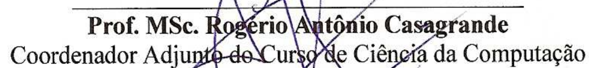
Orientadora: Profª. MSc. Christine Vieira Scarpato

CRICÍUMA, JULHO DE 2009

JOELSON PERDONÁ SERAFIN

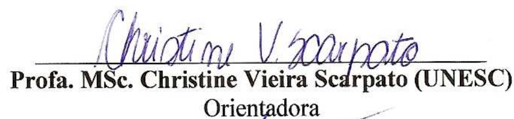
MANIPULAÇÃO DE AUTÔMATOS FINITOS NO AFLAB

Submetido ao corpo docente do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense como um dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

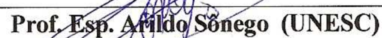


Prof. MSc. Rogério Antônio Casagrande
Coordenador Adjunto do Curso de Ciência da Computação

Banca Examinadora:



Profa. MSc. Christine Vieira Scarpato (UNESC)
Orientadora



Prof. Esp. Arildo Sônego (UNESC)

Prof. MSc. Gustavo Bisognin (UNESC)

Para minha esposa Márcia Mattei Serafim e filha Letícia Mattei Serafim.

AGRADECIMENTOS

Agradeço primeiramente a Deus, que me permitiu mais essa vitória iluminando todos os passos da minha vida.

À minha orientadora, Profª. M.Sc. Christine Vieira Scarpato, pelo tempo disponibilizado na solução de dúvidas e por ter acreditado na minha capacidade de desenvolvimento desse projeto.

À minha esposa, Márcia Mattei Serafim, que sempre me apoio nas minhas decisões e que muito me incentivou para conclusão desse projeto.

Aos meus pais, Jorge Bernadini Serafim e Zuleide Perdoná Serafim que sempre disponibilizaram de recursos e esforços para me proporcionar uma boa educação.

Aos meus amigos e colegas do Curso de Ciência da Computação pelos momentos de alegria compartilhados juntos. A todos que de alguma forma contribuíram para realização deste projeto.

RESUMO

Esse projeto é uma continuação do projeto do AFLAB, onde nesse módulo se desenvolve a minimização de autômatos finitos determinísticos, geração da gramática regular e transformação de autômato finito não determinístico em determinístico. Para alcançar estes objetivos foi realizado um estudo sobre os autômatos finitos determinísticos e não determinísticos, compreendendo seus conceitos, suas características e equivalências. Este projeto ainda aborda as técnicas de minimização de estados de um autômato finito determinístico, a possibilidade de representação de um autômato por meio da gramática e as etapas do processo de determinar um autômato finito não determinístico. Finalizando este trabalho são descritas as características da ferramenta desenvolvida, metodologia utilizada no seu desenvolvimento e os algoritmos que permitem a realização dos objetivos propostos.

Palavras-Chave: Minimização, Geração de Gramática, Transformação, Autômato Finito Determinístico, Autômato Finito Não Determinístico,

ABSTRACT

This is a continuation of the Project AFLAB, which in this module it is developed the minimization of deterministic finite automata, the generation of the regular grammar and the transformation from non-deterministic finite automata into deterministic. In order to achieve these goals, it was carried out a study about deterministic and non-deterministic finite automata, comprehending its concepts, features and equivalents. This project still approaches the minimization techniques of states of a deterministic finite automata, the possibility of the representation of an automata through the grammar and the steps of the process of determining a non-deterministic finite automata. To complete this paper, some features of the developed tool, the methodology used in its development and the algorithms which allow the achievement of the proposed goals are described.

Key words: Minimization; Generation of Grammar; Transformation; Deterministic Finite Automata; Non-deterministic Finite Automata

LISTA DE FIGURAS

Figura 1. Autômato finito como uma máquina com controle finito 21

Figura 2. Representação em diagrama do AFD 22

Figura 3. Representação em tabela de transição do AFD 22

Figura 5. Representação em tabela do AFND... 25

Figura 6. AFLAB – Autômato Finito na Forma Tabular 41

Figura 7. AFLAB – Autômato Finito na Forma Gráfica 42

Figura 8. Diagrama de Caso de Uso da visão do usuário – Módulo de Transformação de AFND em AFD 44

Figura 9. Diagrama de Caso de Uso da visão do sistema – Módulo de Transformação de AFND em AFD 44

Figura 10. Diagrama de Caso de Uso da visão do usuário – Módulo de Minimização de AFD 44

Figura 11. Diagrama de Caso de Uso do visão do sistema – Módulo de Módulo de Minimização AFD 45

Figura 12. Diagrama de Caso de Uso da visão do usuário – Módulo de Geração de GR 45

Figura 13. Diagrama de Caso de Uso da visão do sistema – Módulo de Geração de GR 45

Figura 14. Diagrama de Atividades– Módulo de Transformação de AFND em AFD ... 46

Figura 15. Diagrama de Atividades– Módulo de Minimização do AFD 47

Figura 16. Diagrama de Atividades– Módulo de Geração de GR 48

Figura 17. Estrutura de Armazenamento 50

Figura 18. Tipos declarados 50

Figura 19. Definição do primeiro estado do AFD..51
Figura 20. Identifica se a trasição já foi determinada 52
Figura 21. Inclusão do estado no AFD...53
Figura 22. Inclusão das transições do estado no AFD 54
Figura 23. Eliminação Estados Inacessíveis 56
Figura 24. Eliminação Estados Mortos 57
Figura 25. Declaração da *TClasse*...58
Figura 26. Inserção dos estados finais e não finais 58
Figura 27. Identificação dos estados equivalentes finais 59
Figura 28. Identificação dos estados equivalentes não finais 60
Figura 29. Identificação dos estados equivalentes 61
Figura 30. Gera gramática a partir de um AF 63
Figura 31. Aba minimização AFD 64
Figura 32. Aba minimização AFD renomeando os estados..65
Figura 33. Tela de Geração de GR..65
Figura 34. Tela de transformação de AFND em AFD 66
Figura 35. Tela de transformação do AFND em AFD com estados renomeados...........67

LISTA DE SIGLAS

AF	Autômato Finito
AFD	Autômato Finito Determinístico
AFND	Automato Finito Não Determinístico
GR	Gramática Regular

SUMÁRIO

1 INTRODUÇÃO 13

1.1 OBJETIVO GERAL 15

1.2 OBJETIVOS ESPECÍFICOS 15

1.3 JUSTIFICATIVA 15

1.4 ESTRUTURA DO TRABALHO 16

2 MÁQUINAS DE ESTADOS FINITOS 17

2.1 AUTÔMATOS FINITOS 19

2.1.1 Autômato Finito Determinístico 21

2.1.1.1 Linguagem de um Autômato Finito Determinístico 22

2.1.2 Autômato Finito Não Determinístico 23

2.1.2.1 Linguagem de um Autômato Finito Não Determinístico 25

2.1.3 Equivalência entre Autômato Finito Determinístico e Autômato Finito Não Determinístico 26

2.1.4 Transformação de Autômato Finito Não Determinístico em Autômato Finito Determinístico 27

2.1.5 Minimização de Autômatos Finitos 29

2.1.6 Geração de Gramática Regular a partir de um Autômato Finito Determinístico 33

3 TRABALHOS CORRELATOS 37

4 AFLAB – LABORATÓRIO DE AUTOMATOS FINITOS 39

4.1 MANIPULAÇÃO DE AUTÔMATOS FINITOS NO AFLAB 43

4.2 METODOLOGIA 43

4.3 DESENVOLVIMENTO 49

4.3.1 Módulo de Transformação de AFND em AFD 49

4.3.2 Módulo de Minimização de AFD 55

4.3.3 Módulo de Geração de Gramática Regular a partir de um Autômato Finito Determinístico 62

4.4 RESULTADOS OBTIDOS 63

CONCLUSÃO 68

REFERÊNCIAS 70

1 INTRODUÇÃO

A disciplina de linguagens formais é tradicionalmente, ensinada sem ajuda de ferramenta de software, restringindo-se ao uso de autômatos finitos com poucos estados, pois a resolução dos teoremas aplicados aos autômatos são resolvidos no papel, sendo um inconveniente no curso de Ciência da Computação. Uma possível solução para o problema seria desenvolver uma ferramenta de software que possa auxiliar o aluno no aprendizado de autômatos finitos. Por essas razões, criou-se o grupo de pesquisa de linguagens formais do curso de Ciência da Computação da UNESC que visa o desenvolvimento de uma Shell de autômatos finitos, denominada AFLAB.

A *Shell* AFLAB foi dividida em vários módulos, sendo que a pesquisa aqui proposta compreende a transformação de autômato finito não determinístico em determinístico, minimização de autômato finito determinístico e geração de gramática regular a partir de um autômato finito determinístico.

Autômatos finitos são utilizados nas disciplinas de compiladores, linguagens formais e teoria da computação. Na disciplina de compiladores são utilizados para elaborar o analisador léxico, enquanto que nas disciplinas de linguagens formais e teoria da computação sua finalidade é de reconhecer se uma determinada sentença pertence ou não a linguagem em uso, atuando desta forma como um reconhecedor. Os autômatos finitos são classificados em: autômato finito determinístico (AFD) e autômato finito não determinístico (AFND). O autômato finito determinístico indica que para cada símbolo de entrada existe apenas um estado ao qual o autômato pode transitar a partir do estado atual. Já o autômato finito não determinístico se caracteriza quando para um símbolo de entrada existem vários estados ao qual o autômato pode transitar a partir do estado atual.

O não-determinismo é uma importante generalização dos modelos de máquinas. Porém a facilidade do não-determinístico, em linguagens formais, nem sempre aumenta o poder de reconhecimento de linguagens.

A classe de AFD é equivalente a classe de AFND, sendo possível o processo de transformação de uma classe em outra. Em alguns casos existe maior facilidade em desenvolver um AFND do que um AFD, devido na grande maioria dos casos a solução determinística não ser trivial e resultar em número relativamente grande de estados. Assim pode acontecer em muitos casos, desenvolver inicialmente uma solução não-determinística e transformá-la em determinística aplicando-se o algoritmo de transformação.

Segundo Menezes (2005) a otimização no tempo de processamento do autômato finito determinístico, pode ser alcançada quando é realizada a etapa de minimização do número de estados. A minimização consiste em gerar um autômato finito determinístico equivalente com menor número de estados possíveis eliminando do autômato: estados inacessíveis, equivalentes e mortos.

Os autômatos finitos são reconhecedores de linguagens regulares e essas linguagens regulares são geradas por um formalismo originalmente projetado chamado de Gramática Regular (GR). As gramáticas regulares demonstram por meio de regras como gerar as sentenças da linguagem regular. Então, partindo do pressuposto que o autômato finito reconhece a linguagem regular gerada por uma gramática regular, pode-se dizer que por meio de um autômato finito pode-se descobrir qual gramática que gerou a linguagem reconhecida.

Os estudos desse trabalho será dirigido para implementação do algoritmo de transformação de um AFND em AFD, implementar os algoritmos de minimização de um AFD e do algoritmo que gera uma GR a partir de um AF.

1.1 OBJETIVO GERAL

Desenvolver módulos do AFLAB que manipulam autômatos finitos.

1.2 OBJETIVOS ESPECÍFICOS

Esta pesquisa possui como objetivos específicos:

a) compreender autômato finito determinístico e autômato finito não determinístico;
b) transformar autômato finito não determinístico em autômato finito determinístico;
c) aplicar a minimização em autômato finito determinístico;
d) gerar gramáticas regulares a partir de autômato finito determinístico.

1.3 JUSTIFICATIVA

Devido à necessidade do aluno possuir um maior conhecimento matemático em comparação a outras áreas do curso de Ciência da Computação e de não receberem uma correção imediata quando trabalham com problemas utilizando lápis e papel, faz com que os alunos da disciplina de linguagens formais tenham maior dificuldade em relação as outras áreas do curso. Estes problemas podem ser amenizados se o aluno possuir o auxílio de uma ferramenta de software para possibilitar a manipulação e prática de todos os conhecimentos adquiridos na disciplina.

Esse estudo complementará o módulo do AFLAB, com as seguintes funcionalidades: transformação de autômato finito não determinístico em autômato

finito determinístico, minimização de autômato finito determinístico e geração de gramática regular a partir do autômato finito determinístico. Todas essas etapas citadas são ensinadas aos alunos da disciplina de linguagens formais com poucos recursos de *software*, isso faz com que os alunos realizem essas tarefas com autômatos reduzidos em número de estados, limitando assim o seu aprendizado.

Esta dificuldade de aprendizado pode ser amenizada com o AFLAB que com o passar do tempo está recebendo novas funcionalidades permitindo que os estudantes compreendam melhor os autômatos e seus teoremas.

1.4 ESTRUTURA DO TRABALHO

O Capítulo 2 descreve a máquina de estado finito, e ainda conceitua um AF. Ainda constam nesse capítulo as características do AFD e AFND, citando as etapas de minimização de um AFD, equivalência e transformação de um AFD e AFND finalizando com geração de gramática regular. Já no Capítulo 3 abrange alguns trabalhos correlatos relacionados ao tema de trabalho.

O Capítulo 4 descreve as principais características e funcionalidades da ferramenta, constando a metodologia aplicada no desenvolvimento. Já na conclusão são apresentadas as considerações finais, dificuldades encontradas e sugestões para próximos trabalhos.

2 MÁQUINAS DE ESTADOS FINITOS

As máquinas de estados finitos podem ser conceituadas como máquinas abstratas que capturam as divisões principais de determinada máquina real. Existem muitos exemplos de máquinas reais que são utilizadas no cotidiano que vão desde máquina de vender jornais, máquina de vender refrigerante, compreendendo ainda relógios, elevadores e chegando a programas de computador, como alguns procedimentos de editores e de compiladores. Pode-se afirmar que o computador digital, levando-se em consideração que seu espaço de memória é limitado, pode ser construído por meio de uma máquina de estados finitos (VIEIRA, 2006).

Uma máquina de estado finito apresenta as características de um computador digital atual, onde, sabe-se que as informações internas são armazenadas de forma binária. Em um certo momento do processamento o computador contém determinadas informações armazenadas em sua memória interna definidas por um padrão de dígitos binários, que podem ser denominados de estado do computador. Como é de conhecimento que um computador possui uma quantidade finita de memória, fica evidente que ele possui um número finito de estados que pode assumir. Seu *clock* interno ou relógio interno irá sincronizar suas ações (GERSTING, 1995).

Em cada ciclo do relógio a entrada pode ser lida, o que pode provocar alguma mudança nas posições de memória logo, muda-se o estado da máquina para um novo estado. O que cada novo estado representa depende da entrada e de seu estado anterior, portanto se estes dois fatores forem conhecidos cada alteração será previsível e não casual. Desta forma, após uma seqüência de ciclos do relógio, a máquina irá produzir uma série de saídas em resposta ao conjunto de entradas (GERSTING, 1995).

Segundo Gersting (1995) observando-se que uma máquina de estados possui características semelhantes a um computador, pode-se relacionar as seguintes propriedades da máquina de estados:

a) as operações da máquina são sincronizadas por ciclos discretos do relógio;
b) a máquina atua de forma determinística, ou seja, para seqüência de entrada existe uma resposta completamente previsível;
c) a máquina fornece respostas mediante às entradas;
d) havendo um número finito de estados que a máquina poderá alcançar, a mesma estará em qualquer momento precisamente em um desses estados. O estado que ela vai estar a seguir será o resultado da função aplicada no estado atual e da entrada atual. O estado atual, contudo, depende dos estados e entradas anteriores e assim sucessivamente até se chegar novamente a configuração inicial. Conseqüentemente, o estado da máquina serve como uma espécie de memória das entradas precedentes;
e) a máquina tem a capacidade de produzir saídas. Cada saída é resultado de uma função aplicada em cada estado atual da máquina.

As máquinas de estados finitos podem ser classificadas em dois tipos: os tradutores e os reconhecedores de linguagens. Os tradutores são máquinas que se caracterizam por possuírem entrada e saída, já os reconhecedores são máquinas com apenas duas saídas possíveis (VIEIRA, 2006).

2.1 AUTÔMATOS FINITOS

A história do estudo dos autômatos finitos inicia em 1930 com Alan Turing, que aprofundou seus estudos em uma máquina abstrata que possuía todas as características que compõem os computadores atuais, no que se diz respeito à capacidade de realização de cálculos. Alan Turing pretendia descrever com precisão a capacidade máxima das máquinas, ou seja, o que elas podiam fazer e o que não podiam fazer. Já na década de 40, vários pesquisadores estudaram modelos de máquinas simples que são conhecidas atualmente como autômatos finitos. Os autômatos propostos, com a função originalmente para reproduzir a função do cérebro humano, comportaram-se muito úteis para várias outras finalidades, como *softwares* que projetam e verificam o comportamento de circuitos digitais; analisadores léxicos de compiladores; *softwares* que fazem varredura em corpos de texto, a fim de encontrar ocorrência de palavras entre outros propósitos (HOPCROFT; ULLMAN; MOTWANI, 2002).

Um autômato finito é uma máquina de estados finito capaz de observar apenas um símbolo de cada vez. Assume-se que a máquina esteja no estado inicial quando começa sua operação. A entrada para um autômato finito é uma seqüência de símbolos de um conjunto finito chamado de alfabeto de entrada, que representa o conjunto de símbolos que o autômato consegue processar. Quando um símbolo de entrada é lido, o autômato pode alterar seu estado em função apenas do estado corrente e do símbolo de entrada, ou seja, ocorre uma transição de estado.

Os autômatos finitos podem ser entendidos como um sistema de estados finitos, ou seja, que possuem número limitado e pré-definido de estados, onde constitui um modelo computacional de tipologia seqüencial, que é objeto de estudo na área da computação e informática, principalmente nas disciplinas de linguagens formais,

compiladores, semântica formal e modelos para concorrência. Ainda os autômatos finitos podem ser vistos como reconhecedores de linguagens regulares ou expressões regulares. Sendo um reconhecedor de linguagem um sistema que recebe como entrada uma seqüência de caracteres x, verifica se a seqüência pertence à linguagem em estudo ou não pertence (MENEZES, 2005).

Os autômatos finitos são modelos matemáticos para descrição de tipos particulares de algoritmos. Os autômatos podem ser utilizados para representar o processo de reconhecimento de padrões em cadeias de entrada, com isso auxiliando na construção de sistemas de varreduras (LOUDEN, 2004).

Segundo Menezes (2005) um autômato finito pode ser visto como uma máquina composta, basicamente, de três partes:

a) *fita*: dispositivo de entrada que contem a informação a ser processada, ou seja contém o conjunto de símbolos para serem reconhecidos. A fita é dividida em várias células onde cada uma armazena um símbolo do alfabeto de entrada. Vale ressaltar que esta fita é finita, sendo exatamente do tamanho da sentença a ser processada;
b) *unidade de controle*: reflete o estado corrente da máquina. Possui uma unidade de leitura (cabeça da fita) a qual acessa uma célula da fita de cada vez, sempre iniciando da célula mais a esquerda da fita e movimentando-se exclusivamente para a direita;
c) *programa ou função de transição*: função que comanda as leituras e define o estado corrente da máquina.

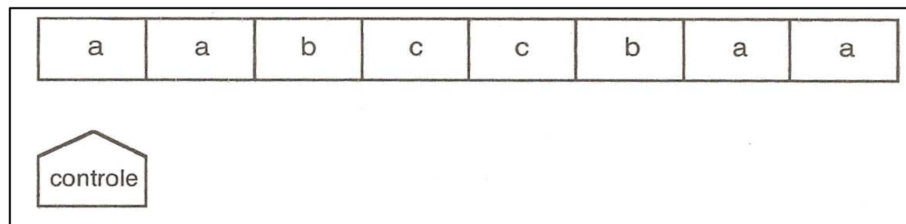


Figura 1. Autômato finito como uma máquina com controle finito
Fonte: MENEZES, P.F.B (2005)

O processo de reconhecimento de um autômato finito pode ser representado por um controle finito, que possui acesso à fita a qual contém a seqüência de símbolos a ser analisada, conforme ilustra a Figura 1. O autômato se movimenta por meio da fita da esquerda para a direita, lendo um símbolo de cada vez. Estando o controle em um estado, a leitura do símbolo de entrada é realizada, ou seja, verifica se o símbolo de entrada pertence ao alfabeto da linguagem. Se o símbolo pertencer ao alfabeto, faz com que o controle passe para outro estado e avance para o próximo símbolo de entrada, caso contrário o processo para no símbolo que não foi reconhecido e restante da seqüência não acontecerá.

Vale ressaltar que um autômato finito não possui memória de trabalho, com isso, para que aconteça o armazenamento das informações necessárias ao processamento, deve-se usar o conceito de estado (MENEZES, 2005).

2.1.1 Autômato Finito Determinístico

O autômato finito determinístico é definido por uma estrutura matemática composta por três unidades: um conjunto de estados, um alfabeto e um conjunto de transições. Os estados são formados por um estado inicial, e por um subconjunto de estados compondo os estados finais (VIEIRA, 2006).

Segundo Louden (2004) um autômato finito determinístico é formalmente definido por uma quíntupla M= (Q, Σ, t, q0, F), onde:

a) Q é o conjunto dos estados possíveis no Autômato o qual é finito;

b) Σ é o alfabeto dos símbolos de entrada;

c) t é a função programa ou função de transição ou simplesmente programa, onde formalmente t: Q x Σ → Q;

d) q0 é o estado inicial do autômato e que pertence ao conjunto Q;

e) F é o conjunto dos estados finais do autômato pertencente ao conjunto Q.

A Figura 2 é representa um AFD em sua forma gráfica e a Figura 3 na forma tabular, onde a e b são os símbolos de entrada, e_o é o estado inicial e e_1 é o estado final do autômato.

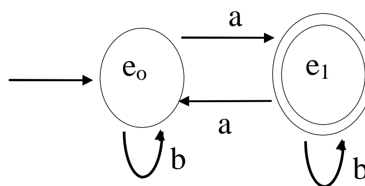


Figura 2. Representação em diagrama do AFD

	t	a	b
→	e_0	e_1	e_0
*	e_1	e_0	e_1

Figura 3. Representação em tabela de transição do AFD

A função de transição possui como argumentos o estado e símbolo de entrada e retorna um estado. Logo, na representação da função t $(e_0,a) = e_1$, se e_0 é um estado e a é um símbolo de entrada, então o estado e_1 será o estado resultante ou estado mapeado pela função (MENEZES, 2005).

2.1.1.1 Linguagem de um Autômato Finito Determinístico

A linguagem reconhecida por um autômato finito determinístico é o conjunto de todas as palavras pertencentes ao alfabeto dos símbolos de entrada aceitos

pelo autômato, incluindo o vazio, partindo de seu estado inicial. Já a linguagem que não é reconhecida (rejeitada) por um autômato finito determinístico é o conjunto de todas as palavras pertencentes ao alfabeto dos símbolos de entrada, incluindo o vazio, e que não são aceitas pelo autômato, a partir de seu estado inicial. Conseqüentemente, cada autômato finito definido sobre o alfabeto do conjunto dos símbolos de entrada, leva a uma participação do conjunto de todas as palavras em duas classes análogas, ou seja, as que aceitam ou as que rejeitam a linguagem. Sendo necessário, quando um dos dois conjuntos for vazio, a participação induzida com apenas um conjunto, que coincide com o conjunto dos símbolos de entrada incluindo o vazio (MENEZES, 2005).

Supondo que J = (Q, Σ, t, q0, F) e a linguagem aceita por J é representada por ACEITA (J) e a linguagem rejeitada por J é representada por REJEITA (J). Baseando-se no conjunto universo dos símbolos de entrada, pode-se afirmar que:

a) ACEITA(J) ∩ REJEITA(J) = ø;

b) ACEITA(J) U REJEITA(J) = $\Sigma^{*;}$

c) ~ACEITA(J) = REJEITA(J);

d) ~REJEITA(J) = ACEITA(J).

2.1.2 Autômato Finito Não Determinístico

Segundo Vieira (2006) os autômatos finitos não determinísticos possuem os mesmos componentes que os autômatos finitos determinísticos, diferenciando no aspecto que os autômatos finitos não determinísticos podem possuir várias transições partindo de um estado com o mesmo símbolo de entrada.

Uma forma comum de compreender o AFND é afirmar que em cada ponto de indecisão, ou seja, quando o autômato finito não sabe para qual estado deve se

dirigir, o autômato escolhe alguma transição, para ter sucesso no reconhecimento da sentença. Se não houver nenhuma transição que leve ao reconhecimento da palavra, o autômato escolhe qualquer uma das transições, partindo do pressuposto que todas levam à rejeição (VIEIRA, 2006).

Uma outra interpretação de não determinismo é descrita por Menezes (2005, p. 55):

> "Visto como uma máquina composta por fita, unidade de controle e programa, pode-se afirmar que um Autômato Finito Não-Determinístico assume um conjunto de estados alternativos, como se houvesse uma multiplicação da unidade de controle, uma para cada alternativa, processando independentemente, sem compartilhar recursos com as demais. Assim, o processamento de um caminho não influi no estado, símbolo lido e posição da cabeça dos demais caminhos alternativos."

Os autômatos finitos não determinísticos são compostos formalmente por uma quíntupla M= (Q, Σ, t, q0, F), sendo:

a) Q é o conjunto dos estados possíveis no autômato o qual é finito;
b) Σ é o alfabeto dos símbolos de entrada;
c) t é a função programa ou função de transição ou simplesmente programa, onde formalmente t: $Q \times \Sigma \rightarrow 2^Q$, indicando que a aplicação da função poderá resultar em mais de uma saída;
d) q0 sendo o estado inicial do autômato e que pertence ao conjunto Q;
e) F é o conjunto dos estados finais do Autômato pertencente ao conjunto Q.

A Figura 4 representa um AFND na forma gráfica e a Figura 5 autômato representa na forma tabular, onde a e b são os símbolos lido pelo autômato que pertencem ao conjunto Σ, e_o é o estado inicial e e_3 é o estado final do autômato. No exemplo abaixo o estado e_o que por meio do símbolo de entrada *a* poderá seguir o estado e_1 ou ficar nele próprio.

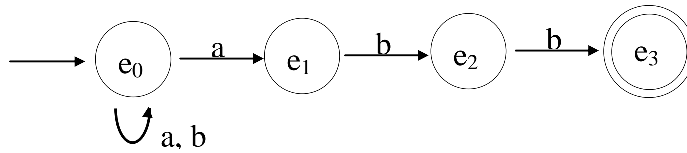


Figura 4. Representação em diagrama do AFND

	δ	A	b
→	e_0	e_0, e_1	e_0
	e_1	-	e_2
	e_2	-	e_3
*	e_3	-	-

Figura 5. Representação em tabela do AFND

Tendo em vista que um autômato finito não determinístico tem a possibilidade de estar em mais de um estado ao mesmo instante de tempo, essa capacidade é explorada na sua utilização durante a busca de certas seqüências de caracteres em longos *strings* de texto (HOPCROFT; ULLMAN; MOTWANI, 2002).

2.1.2.1 Linguagem de um Autômato Finito Não Determinístico

A linguagem aceita por um AFND é composta por um conjunto de palavras pertencentes ao alfabeto de símbolos de entrada, que por meio de uma das transições alternativas pode se aceitar a palavra. Já a linguagem rejeitada pelo AFND é o conjunto de palavras pertencentes ao alfabeto de símbolos de entrada, que é rejeitada por meio das transições alternativas (MENEZES, 2005).

2.1.3 Equivalência entre Autômato Finito Determinístico e Autômato Finito Não Determinístico

Um AFD é equivalente ao AFND. A comprovação dessa afirmação é que por meio de qualquer AFND é possível construir um AFD equivalente, que reconheça a mesma linguagem. A idéia principal é que a partir de um AFD se simule as equivalentes combinações dos estados alternativos contidos no AFND (MENEZES, 2005).

A descrição de várias linguagens é facilitada quando utiliza-se o AFND, em comparação quando utiliza-se o AFD. Pode-se ainda afirmar que toda linguagem descrita por um AFND também pode ser descrita por um AFD. Além desse fato, o AFD tem tanto estados quanto o AFND correspondente e possui com freqüência maior número de transições. Porém, em casos extremos o menor AFD pode possuir 2^n estados, enquanto o menor AFND possui apenas n estados, levando em consideração o reconhecimento da mesma linguagem (HOPCROFT; ULLMAN; MOTWANI, 2002).

Segundo Menezes (2005) para comprovar que AFD representado por M_D simula as computações do AFND representado por M segue o teorema abaixo:

a) indução no tamanho da palavra;
b) mostrar que $t_D*[(q_0), w] = (q_1...q_u)$ se $t*(\{ q_0 \}, w) = \{ q_1,.., q_u \}$
   - base de indução. . $|w| = 0$. Portanto $w = \varepsilon$:

     $t_D *[(q_0), \varepsilon] = (q_0)$ se e somente se $t*(\{ q_0 \}, \varepsilon) = \{ q_0 \}$

     verdadeiro, por definição de computação;
   - hipótese de indução $|w| = n$ e $n \geq 1$. Suponha que:

     $t_D*[(q_0), w] = (q_1 \ldots q_u)$ se e somente se $t*(\{ q_0 \}, w) = \{ q_1, \ldots, q_u\}$;
   - passo: $|wa| = n + 1, n \geq 1$,

     $t_D*[(q_0, w_a] = (p_1 \ldots p_v)$ se somente se $t*(\{ q_0 \}, w_a) = \{ p_1, ..., p_v \}$

- equivale (hipótese de indução)

  $t_D[(q_1 \ldots q_u), a] = (p1 \ldots pv)$ se e somente se $t^*(\{ q_1, \ldots, q_u \}, a) = \{ p_1, \ldots, p_v \}$

- verdadeiro, por definição de t_D

Logo, M_D simula M para qualquer entrada w pertencente a Σ*.

2.1.4 Transformação de Autômato Finito Não Determinístico em Autômato Finito Determinístico

Em alguns casos existe dificuldade na aplicação de uma solução determinística para o reconhecimento de linguagens, devido ao grande número de estados gerado com esse tipo de solução. Já a solução não determinística se apresenta mais simples e com poucos estados, por isso em muitos casos se desenvolve uma solução não determinística e posteriormente se realiza a transformação para solução determinística (MENEZES,2005).

A transformação de AFND para um AFD consiste em a partir de um AFND eliminar transições múltiplas de um estado com o mesmo símbolo de entrada, o que caracteriza esse autômato, com o intuito do AFD reconhecer sentenças equivalentes as reconhecidas pelo AFND.

Supondo que L é um conjunto de símbolos aceito por um AFND, então seja M= (Q, Σ, t, q0, F) um AFND pode construir um AFD M’=(Q’, Σ’, t’, q0’, F’) seguindo o teorema abaixo (SCARPATO,2004):

a) Q’ = {ρ(Q)} - isto é, cada estado de M’ será um subconjunto de estados de M;
b) qo’ = [qo] - ou seja, qo’ será o ρ(Q) composto apenas por q0.

c) $F' = \{\rho(Q) \mid \rho(Q) \cap F \neq \varphi\}$;

d) para cada $\rho(Q) \subset Q'$

definimos $t'(\rho(Q),a) = \rho'(Q)$,

onde $\rho'(Q) = \{p \mid \text{para algum } q \in \rho(Q), t(q, a) = p\}$;

ou seja, se $\rho(Q) = [q1, q2, ..., qr] \in Q'$ e se

$t(q1,a) = p1, p2, ..., pj$

$t(q2,a) = pj+1, pj+2, ..., pk$

: : : :

$t(qr, a) = pi, pi+1, ..., pn$ são as transições de M,

então $\rho(Q) = [p1, ..., pj, pj+1, ..., pr, pi, ..., pn]$ será um estado de M',

e M' conterá a transição: $t'(\rho(Q),a) = \rho'(Q)$.

A seguir tem-se uma breve explanação sobre o processo de transformação de AFND para AFD (SCARPATO,2004):

a) determinar a primeira linha da tabela de transição do AFD como sendo o conjunto unitário contendo apenas o estado inicial do AFND;
b) verificar, para cada estado do AFD, se as transições foram determinadas, ou seja, identificar se o rótulo K, possivelmente os rótulos de vários estados do AFND representando a união dos mesmos, de cada transição já é usado como rótulo de algum estado (em alguma linha) da tabela de transição do AFD. Em caso negativo, criar uma nova linha contendo aquele rótulo K e determinar, para cada símbolo de entrada do alfabeto Σ, o conjunto resultante da união dos vários estados que compõem K. Repetir o segundo passo até que todas as transições tenham sido determinadas;

c) determinar o estado inicial do AFD como sendo a primeira linha da tabela de transições;

d) determinar o(s) estado(s) final(is) do AFD como sendo todos os estados (todas as linha) rotuladas com conjuntos que contenham pelo menos um estado final do AFND.

2.1.5 Minimização de Autômatos Finitos

A eficiência no processamento é extremamente importante no reconhecimento de linguagens. Com o objetivo de se conseguir uma maior eficiência do autômato finito determinístico se realiza a minimização, onde a partir de um AFD original existe um AFD equivalente, mas com menor número de estados, sem alterar o reconhecimento da linguagem, ou seja, a mesma entrada reconhecida pelo autômato original será reconhecida pelo autômato minimizado.

O autômato finito minimizado é único, sendo assim diferentes autômatos finitos que reconhecem a mesma linguagem, ao serem minimizados geram o mesmo autômato finito, apenas diferenciando na identificação dos estados (MENEZES, 2005).

O processo de minimização consiste na eliminação de alguns estados que possuem as seguintes características (FURTADO, 2002):

a) estados inacessíveis: um estado $q \in Q$ é inacessível (ou inalcançável) quando não existe w_1 tal que a partir de qo, q seja alcançado, ou seja, não existe $w_1 \mid t(qo, w_1) = q$, onde w_1 é uma sentença ou parte dela;

b) estado inútil ou morto: um estado $q \in Q$ é inútil se ele $\notin F \wedge \exists w_1 \mid t(q, w1) = p$, onde $p \in F \wedge w_1$ é uma sentença ou parte dela, ou seja, q é

morto se ele não é final e a partir dele nenhum estado final pode ser alcançado.

c) estados equivalentes: um conjunto de estados q1, q2, ..., qj são equivalentes entre sí, se eles pertencem a uma mesma classe de equivalência. Sendo uma classe equivalente um conjunto de estados q1, q2, ..., qj está em uma mesma classe de equivalência se t (q1, a), t (q2, a), ..., t (qj, a), para cada a ∈ ∑, resultam respectivamente nos estados qi, qi+1, ..., qn, e estes pertencem a uma mesma classe de equivalência.

Segundo Menezes (2005) para um autômato finito a ser minimizado deve atender aos seguintes pré-requisitos:

a) o autômato deve ser determinístico;

b) todos estados do autômato devem ser alcançáveis a partir do estado inicial;

c) a função de transição deve ser total, ou seja, partindo de qualquer estado, possui transições para todos os símbolos do alfabeto.

A minimização do autômato finito deve-se seguir o algoritmo abaixo:

a) entrada. A.F.D. M = (Q, Σ, t, qo, F);

b) saída. A.F.D. Mínimo M’ = (Q’, Σ, t’, qo’, F’) | M’ ≡ M;

c) método.

- elimine os estados Inacessíveis;
- elimine os estados Mortos;
- construa todas as possíveis classes de equivalência de M.
- construa M’, como segue:

1. Q': é o conjunto de classe de equivalência obtida;
2. qo': é a classe de equivalência que contem qo;
3. F': é o conjunto das classes de equivalência que contenham pelo menos um elemento $\in$ F; ou seja : {[q] | $\exists$ p $\in$ F em [q], onde [q] é uma classe de equivalência};
4. t' - t'([p], a) = [q] $\Leftrightarrow$ t (p_1, a) = q_1 é uma transição de M $\wedge$ p_1 e q_1 são elementos de [p] e [q] respectivamente.

Segundo Menezes (2005) para se realizar a eliminação dos estados equivalentes do autômato finito deve se seguir os passos:

a) construção da tabela verdade: construir uma tabela com os estados presentes no autômato;

b) marcação de estados trivialmente não equivalentes: na tabela construída no passo anterior, o estado inicial e os estados finais;

c) marcação de estados não equivalentes: sendo cada par {qu, qv) não marcados e que $a \in$ Q, imagina-se que t (qu, a) = pu e t (qv, a) = pv então:

- se pu=pv, então qu é igual a qv, para o símbolo de entrada *a* por isso não deve ser marcado;

- se pu $\neq$ pv e o par {pu, pv} não está marcado, então {qu, qv}, será incluído a partir de {pu, pv} em outra lista para uma posterior análise. Caso contrário de se pu $\neq$ pv e o par {pu, pv} está marcado, então {qu, qv} não são equivalentes e se {qu, qv} inicia um lista de pares, então deve-se marcar todos os pares. Deve-se repetir o

processo recursivamente para os outros os estados da lista, ou seja, se algum par de estados inicia uma lista, também deve ser marcado.

d) unificação dos estados equivalentes. Os estados dos pares que não receberão marcação são equivalentes e podem ser unidos como segue:

- a equivalência de estados é transitiva;
- estados não-finais e finais podem ser transformados em um único estado não-final e final respectivamente;
- caso algum dos estados equivalentes é inicial, obrigatoriamente o estado unificado também deve ser inicial;
- devem se preservar todas as transições com origem em um estado equivalente, mas passam a ter origem no estado unificado.

A eliminação dos estados inúteis ou mortos acontece da seguinte forma:

a) marcar inicialmente os estados finais;
b) marcar todos os estados e_k que alcançam um estado marcado e_i por uma transição com qualquer símbolo de entrada;
c) eliminar os estados não marcados, deixando indefinidas as transições que levam a um estado eliminado.

A eliminação dos estados inacessíveis segue o seguinte procedimento:

a) marcar primeiramente o estado inicial;
b) para cada estado e_i marcado, marcar todos os estados e_k que podem ser alcançados por uma transição a partir de e_i;
c) eliminar os estados não marcados.

Ao fim do processo descrito no algoritmo, para verificar se o autômato chegou ao limite de minimização, aplica-se avaliação dos pré-requisitos de minimização, e com isso se analisa se o processo foi realizado com sucesso.

2.1.6 Geração de Gramática Regular a partir de um Autômato Finito Determinístico

Segundo Vieira (2006) uma gramática G é definida como sendo uma quádrupla G = (V_N, V_T, P, S), onde:

a) V_N é um conjunto finito de símbolos denominados símbolos não-terminais, usados na descrição da linguagem;

b) V_T é um conjunto finito de símbolos denominados símbolos terminais, os quais são os símbolos propriamente ditos;

c) P é conjunto finito de pares (α, β) denominados regras de produção (ou regras gramaticais) que relacionam os símbolos terminais e não-terminais;

d) S é o símbolo inicial da gramática pertencente a V_N, a partir do qual as sentenças de uma linguagem podem ser geradas.

As regras gramaticais são representadas por $\alpha ::= \beta$ ou $\alpha \rightarrow \beta$, onde α e β são sentenças sobre V, com α envolvendo pelo menos um símbolo pertencente a V_N. Uma seqüência de regras de produção da forma $\alpha \rightarrow \beta_1$, $\alpha \rightarrow \beta_2$, ... $\alpha \rightarrow \beta_n$, ou seja, com a mesma componente do lado esquerdo, pode ser abreviada como uma única produção na forma: $\alpha \rightarrow \beta_1 \mid \beta_2 \mid ... \mid \beta_n$. O significado de uma regra de produção $\alpha \rightarrow \beta$ é α produz β ou α é definido por β (MENEZES, 2002).

Os autômatos finitos são reconhecedores de linguagens regulares e essas linguagens regulares são geradas por um formalismo originalmente projetado chamado de gramática regular. As gramáticas regulares demonstram por meio de regras como gerar as sentenças da linguagem regular. Então partindo do pressuposto que o autômato finito reconhece a linguagem regular gerada por uma gramática regular, pode-se dizer que por meio de um autômato finito descobri-se qual gramática que gerou a linguagem reconhecida. Devido a esta situação a transformação de autômato finito determinístico em gramática regular será um dos objetos de estudo.

Se M = ($Q; \Sigma; t; q_0; F$) é um AFD então existe uma gramática regular G = ($V_N; V_T; P; S$), tal que L(G) = T(M). Para obter G:

a) $V_N = Q$;

b) $V_T = \sum$;

c) $S = q_0$;

d) definindo P:

- se t(B, a) = C então adicione $B \rightarrow aC$ em P;
- se t(B, a) = $C \wedge C \in F$ então adicione $B \rightarrow a$ em P;
- se $q_o \in F$, então $\varepsilon \in T(M)$. Neste caso, L(G) = T(M) – {ε}, G_1 | $L(G_1)$ = L(G) U {ε} e portanto, $L(G_1)$ será igual a T(M);

  Senão, $\varepsilon \notin T(M)$ e L(G) = T(M).

Para demonstrar que L(G) = T(M), deve-se mostrar que:

a) $T(M) \subseteq L(G)$, para mostrar que T(M) está contido em L(G), significa mostrar que, se T(M) contém x, então $x \in L(G)$ então:

- seja $x = a_1a_2...a_n$ uma seqüência aceita por M;

- se M aceita x, então $\exists$ uma seqüência de estados S, A_1, A_2, ..., A_{n-1}, A| $t(S, a_1) = A_1$, $t(A_1, a_2) = A_2$... $t(A_{n-1}, a_n) = A$, onde S é o estado inicial e A é um estado final. Por definição, para que essas transições existam, G deverá possuir as seguintes produções:

  $S \rightarrow a_1A_1$, $A_1 \rightarrow a_2A_2$, ... $An\text{-}_1 \rightarrow na$

  e, se essas produções existem, então

  $S \Rightarrow a_1A_1 \Rightarrow a_1a_2A_2 \Rightarrow ... \Rightarrow a_1a_2...na_{-1}A_{n-1} \Rightarrow a_1a_2... a_n$ é uma derivação em G;

  logo, como $x = a_1a_2...a_n \wedge S \rightarrow x$, então $x \in L(G)$;

- se $\varepsilon \in T(M)$. Neste caso, por definição, $S \in F$ e se $S \in F$ é porque $S \rightarrow \varepsilon \in P$, logo $\varepsilon \in L(G)$;

  assim, **T(M) ⊆ L(G).**

b) $L(G) \subseteq T(M)$, mostrar que L(G) está contido em T(M), significa mostrar que se $x \in L(G)$ então T(M) contém x, ou seja, M aceita x, então:

- seja $x = a_1a_2...a_n \in L(G)$;
- se $x \in L(G)$, então $S \rightarrow x$;
- como G é uma G.R., a derivação $S \rightarrow x$ é da forma

  $S \Rightarrow a_1A_1 \Rightarrow a_1a_2A_2 \Rightarrow ... \Rightarrow a_1a_2...na_{-1}A_{n-1} \Rightarrow a_1a_2...a_n$.

  Logo,

  $S \rightarrow a_1A_1$, $A_1 \rightarrow a_2A_2$, ... $An\text{-}_1 \rightarrow a_n$ são produções de G;

- assim sendo, por definição, M consistirá das seguitnes transições:

$t(S,a_1) = A_1$, $t(A_1,a_2) = A_2$, ... $t(A_{n-1}, a_n) = A$

- portanto, como $x = a_1a_2...a_n$, $t(S, x) = A \wedge A \in F$, conclui-se que T(M) contém x.
- se $\varepsilon \in L(G)$ é porque $S \rightarrow \varepsilon \in P$. Neste caso, por definição, $S \in F$ e, portanto, T(M) contém ε. Logo, **L(G) ⊆ T(M).**

3 TRABALHOS CORRELATOS

O estudo sobre autômatos finitos vem sendo desenvolvido em várias pesquisas. A seguir são relacionadas algumas pesquisas que foram realizadas sobre o tema desse trabalho:

a) na Universidade Federal de Minas Gerais (UFMG) tem-se o desenvolvimento de um software em linguagem Java, denominado *Language Emulator* como trabalho no curso de pós-graduação. Esse sistema permite o usuário a manipulação de autômatos finitos determinístico e não determinístico , assim como, a minimização de AFD e transformação de AFND em AFD (VIEIRA; VIEIRA, 2002);
b) originado de um trabalho de especialização do Departamento de Ciências da Computação da Universidade Federal de Lavras (UFLA) e desenvolvida em linguagem Java, a ferramenta GAM (*Ginux Abstract Machine*) pode ser utilizada em ambientes GNU/Linux, Windows e em sistemas operacionais que suportem GTK e GTKmm. Ela permite ao usuário, dentre outras funcionalidades, as operações de minimização e transformação de AFND em AFD (JUKEMURA; NASCIMENTO; UCHOA, 2005).
c) desenvolvido na Universidade de São Francisco (USFCA), utilizando a linguagem de programação Java, a ferramenta VAS (*Visual Automata Simulator*), possibilita ao usuário a transformação de AFND em AFD, além de outras funcionalidades (BOVET, 2008).
d) já na Universidade Regional de Blumenau (FURB) desenvolveu-se, por meio de um trabalho de disciplina de Linguagens Formais, a Ferramenta

para a Transformação de Autômato Finito Não Determinístico em Autômato Finito Determinístico com fim especifico de transformação de AFND em AFD, criada no ambiente de programação *Delphi* (SANTOS; VARGAS, 2004)**.**

e) iniciado o desenvolvimento no *Rensselaer Polytechnic Institute*, USA, por volta de 1990 e continuado posteriormente em 1994 pela Duke University, o *Java Formal Language and Automata Package* (JFLAB) é um *software* de *interface* gráfica que permite criar e simular diversos tipos de autômatos e fazer conversões entre diferentes representações de linguagens, tais como autômato finito para gramática regular, gramática regular para autômato de pilha, expressão regular para autômato finito, autômato finito para expressão regular entre outras funcionalidades (RODGER, 1990).

4 AFLAB – LABORATÓRIO DE AUTOMATOS FINITOS

O AFLAB é um ambiente de criação e manipulação de AF, que surgiu da necessidade de um *software* que auxilie no aprendizado dos alunos nas disciplinas de Linguagens Formais, Compiladores e Teoria da Computação do curso de Ciência da Computação da UNESC.

O módulo inicial desenvolvido por Gaidzinski (2007), realiza o reconhecimentos de AFD ou AFND representados na forma tabular ou forma gráfica, para posteriormente realizar o reconhecimento de sentenças.

O desenvolvimento do AFLAB foi realizado em linguagem *pascal*, utilizando a plataforma Borland® Delphi™ *Enterprise*, versão 7.0. A escolha desse ambiente de programação é devido ao fato de já ser utilizado no meio acadêmico, permitindo que o código seja estudado ou alterado conforme necessidade do acadêmico, além da característica de gerar programas bastante leves (GAIDZINSKI, 2007).

A ferramenta possui uma estrutura única para o armazenamento de autômatos finitos, sem a restrição em relação ao autômato ser determinístico ou não determinístico, representado na forma gráfica ou tabular, permitindo com que o autômato finito construído na forma tabular seja visualizado também na forma gráfica e vice versa. Essa estrutura foi dividida basicamente em três classes: *Testados, Talfabeto e TConexao*. Sendo que na classe *Testados* é armazenado o nome de cada estado pertencente ao AF, além conter alguns atributos do estado, como se é um estado inicial ou estado final e as coordenadas responsáveis pela posição para cada estado construído em forma gráfica. Já a classe *Talfabeto* armazena os símbolos de entrada do AF. Finalizando a *Tconexao* tem a função de armazenar as informações referentes às transições pertencente ao AF, contendo nessa classe três atributos: estado origem

(estados de origina a transição), símbolo de entrada (pertencente a classe *Talfabeto*) e estado de destino (estado a qual a transição se destina) (GAIDZINSKI, 2007).

Ao iniciar a utilização da ferramenta o usuário pode digitar um novo autômato (na forma gráfica ou tabular) ou abrir arquivo salvo anteriormente.

No módulo Forma Tabular o usuário deve preencher os estados do AF, seus símbolos de entrada, o estado inicial e o conjunto dos estados finais. Após o preenchimento dessas informações deve-se definir as transições de cada estado. Posteriormente a sentença a ser testada pode ser informada no campo correspondente, que trará como resposta a palavra "VERDADEIRA" se tal sentença for reconhecida pelo autômato informado ou "FALSA" caso contrário. Esse módulo ainda possui a opção de migração do autômato finito construído na Forma Tabular para ser visualizado na sua forma gráfica, e possui a opção de exclusão do AF. O módulo de Forma Tabular pode ser observado na Figura 6.

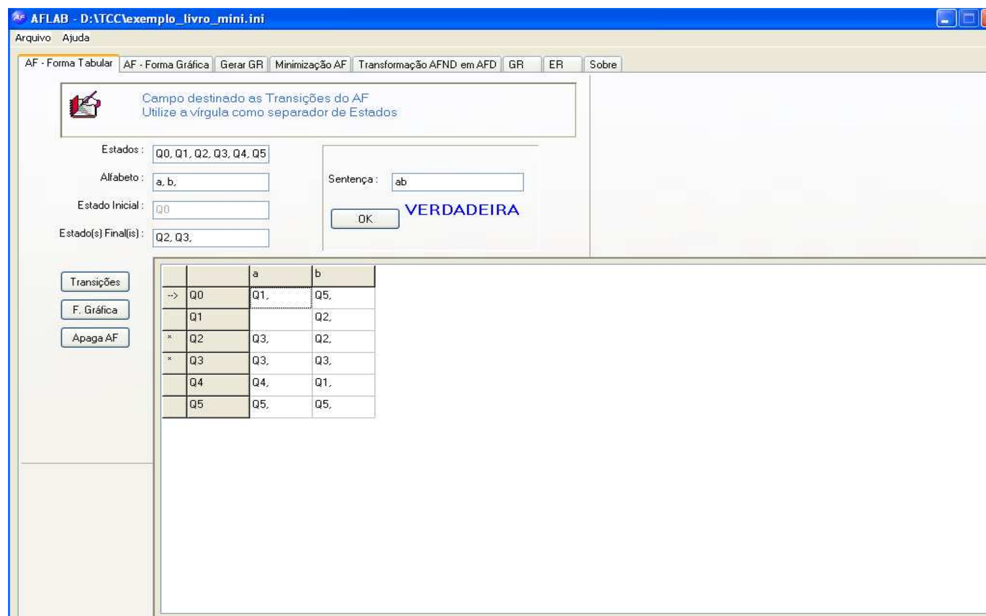


Figura 6. AFLAB – Autômato Finito na Forma Tabular
Fonte: GAIDZINSKI, M. (2007)

O módulo de Forma Gráfica do AFLAB permite as mesmas funcionalidades que estão presentes na Forma Tabular, diferenciando que na Forma Gráfica o AF é desenhado na tela. O módulo de Forma Gráfica pode ser observado na Figura 7.

Figura 7. AFLAB – Autômato Finito na Forma Gráfica
Fonte: GAIDZINSKI, M. (2007)

Vale ressaltar que além desse módulo inicial do AFLAB, foi desenvolvido por Aline Teixeira (2008) como Trabalho de Conclusão de Curso de Ciências da Computação da UNESC, o módulo de transformação de expressões regulares em AF e o módulo para obtenção de AF a partir de GR. Ainda no mesmo ano, Marlon de Matos de Oliveira (2008) também como Trabalho de Conclusão de Curso de Ciências da Computação da UNESC, criou os módulos que permitem utilizar AFD e transformá-lo em uma máquina de Mealy ou de Moore, para simulação de saídas em Textos, Imagens ou Sons, por meio de uma sentença válida.

Verificando-se a possibilidade de aperfeiçoamento ainda mais da ferramenta AFLAB, foram iniciados os módulos de minimização do AF, transformação de AFND em AFD e geração de GR.

4.1 MANIPULAÇÃO DE AUTÔMATOS FINITOS NO AFLAB

O módulo de transformação de AFND em AFD possui o algoritmo que realiza a conversão do autômato finito digitado pelo usuário e a partir desse AFD gerado tem a opção de minimizar os estados do AF em questão, sendo esse o segundo módulo. Ainda como terceiro módulo o usuário poderá gerar a gramática do autômato inicialmente digitado.

Os três módulos além de utilizarem as classes já existentes de autômatos finitos da versão inicial do AFLAB no que diz respeito aos estados, alfabeto e conexões, utilizam novas classes que foram desenvolvidas para o módulo de transformação de AFND em AFD e minimização de AFD. Porém como foram mantidas as classes principais do módulo inicial, o AFLAB possibilita integração de novos módulos.

4.2 METODOLOGIA

Para realização do desenvolvimento dos novos módulos do AFLAB, foi realizado um levantamento bibliográfico sobre os AFD e AFND, verificando seus conceitos e características. Além do estudo de algoritmos que realizam a etapa de transformação do AFND em AFD, minimização do AFD e geração de gramática regular. Após esse estudo realizou-se a modelagem por meio de UML para facilitar o entendimento das funcionalidades e operações dos módulos desenvolvidos. As Figuras 8 e 9 mostram os diagramas de Caso de Uso referente ao módulo de transformação de um AFND em AFD, já as Figuras 10 e 11 representam o módulo de minimização de um AFD e as Figuras 12 e 13 apresentam os diagramas de Caso de Uso da geração de GR a

partir de um AFD. Com este tipo de diagrama é possível visualizar a função de cada entidade pode exercer no sistema.

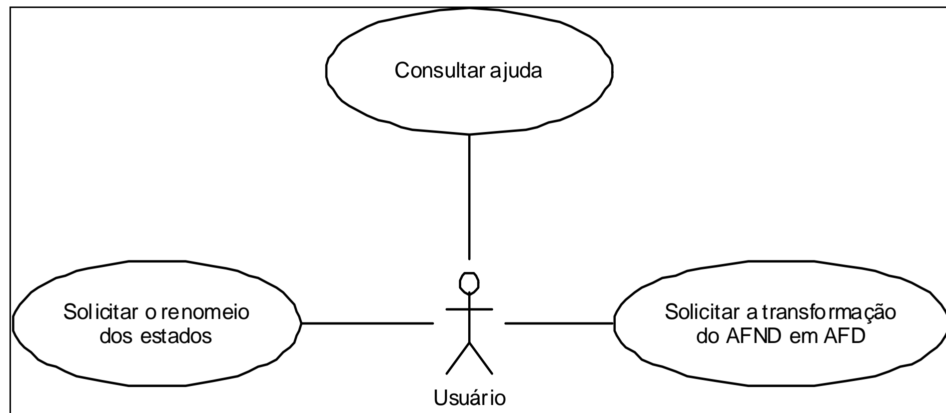


Figura 8. Diagrama de Caso de Uso da visão do usuário – Módulo de Transformação de AFND em AFD

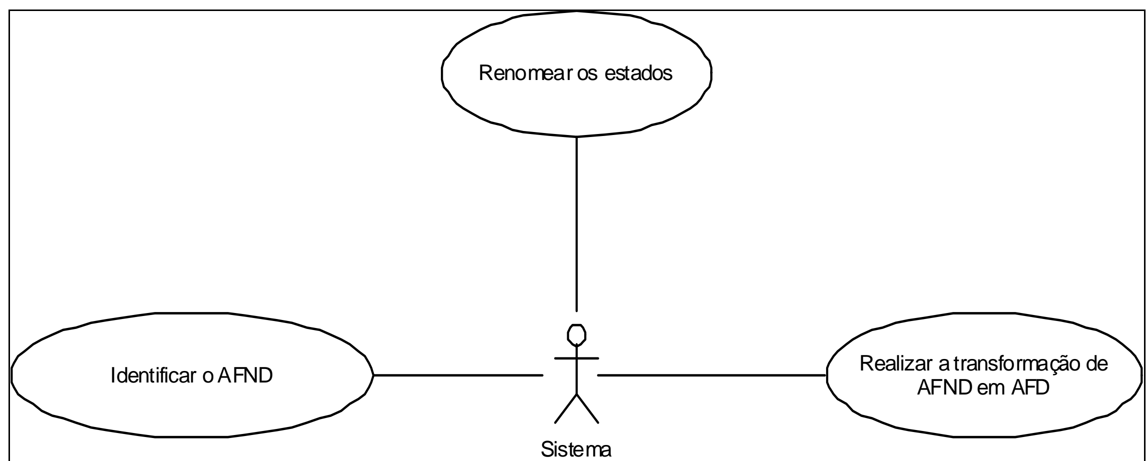


Figura 9. Diagrama de Caso de Uso da visão do sistema – Módulo de Transformação de AFND em AFD

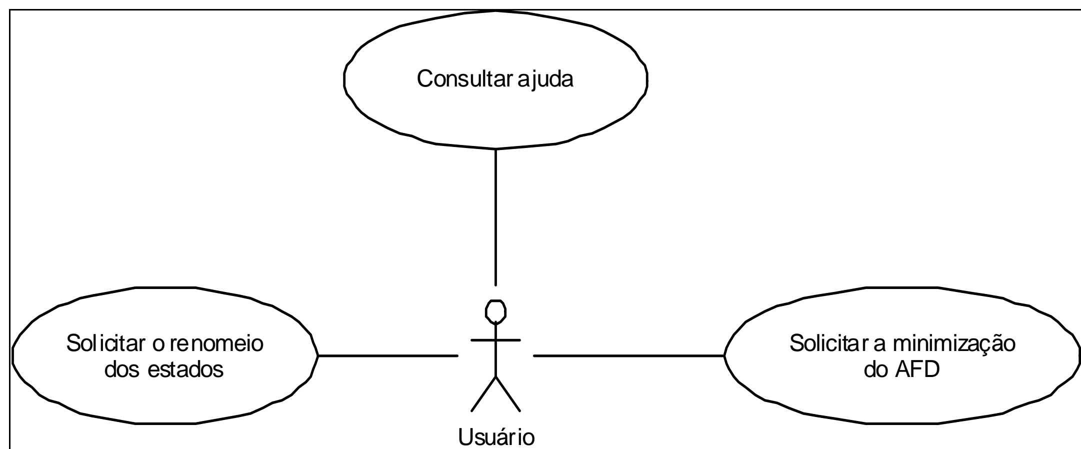


Figura 10. Diagrama de Caso de Uso da visão do usuário – Módulo de Minimização de AFD

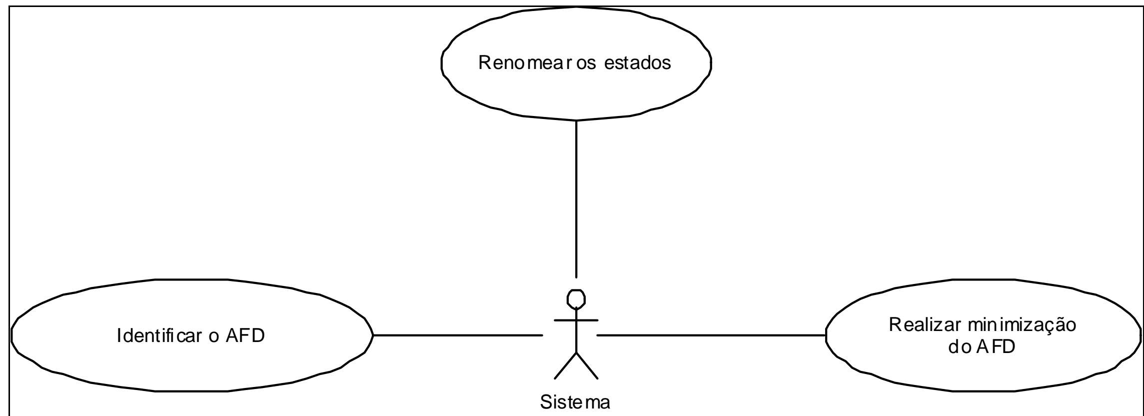


Figura 11. Diagrama de Caso de Uso da visão do sistema – Módulo de Módulo de Minimização AFD

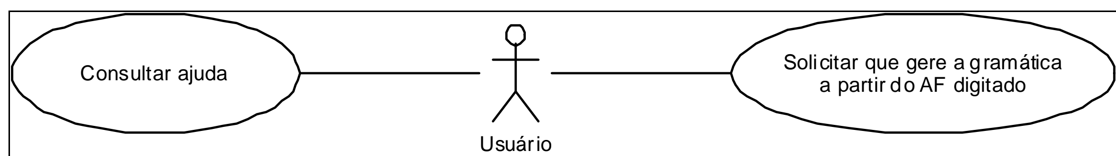


Figura 12. Diagrama de Caso de Uso da visão do usuário – Módulo de Geração de GR

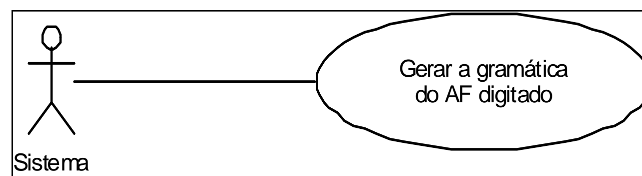


Figura 13. Diagrama de Caso de Uso da visão do sistema – Módulo de Geração de GR

Já o diagrama de Atividades apresentado nas figuras 14, 15 e 16, apresenta o fluxo de cada módulo.

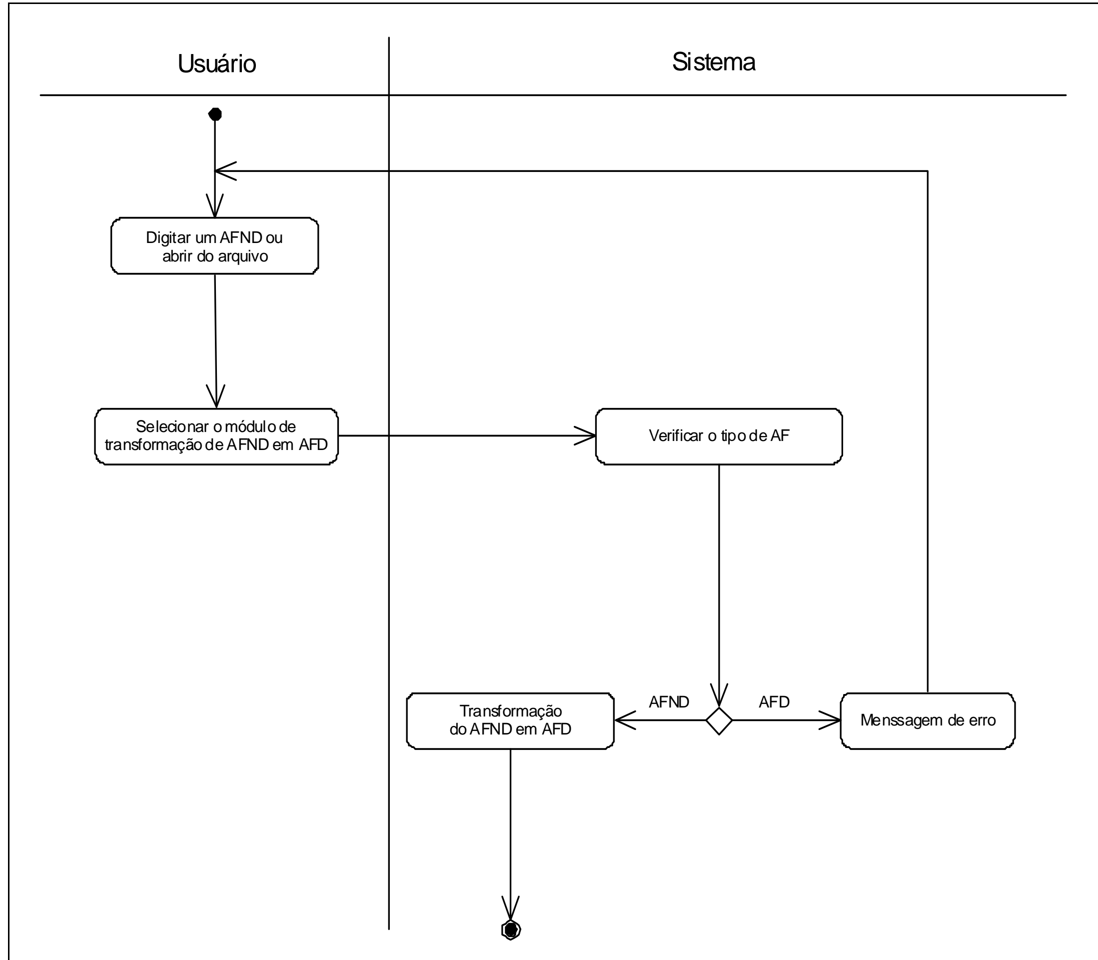


Figura 14. Diagrama de Atividades– Módulo de Transformação de AFND em AFD

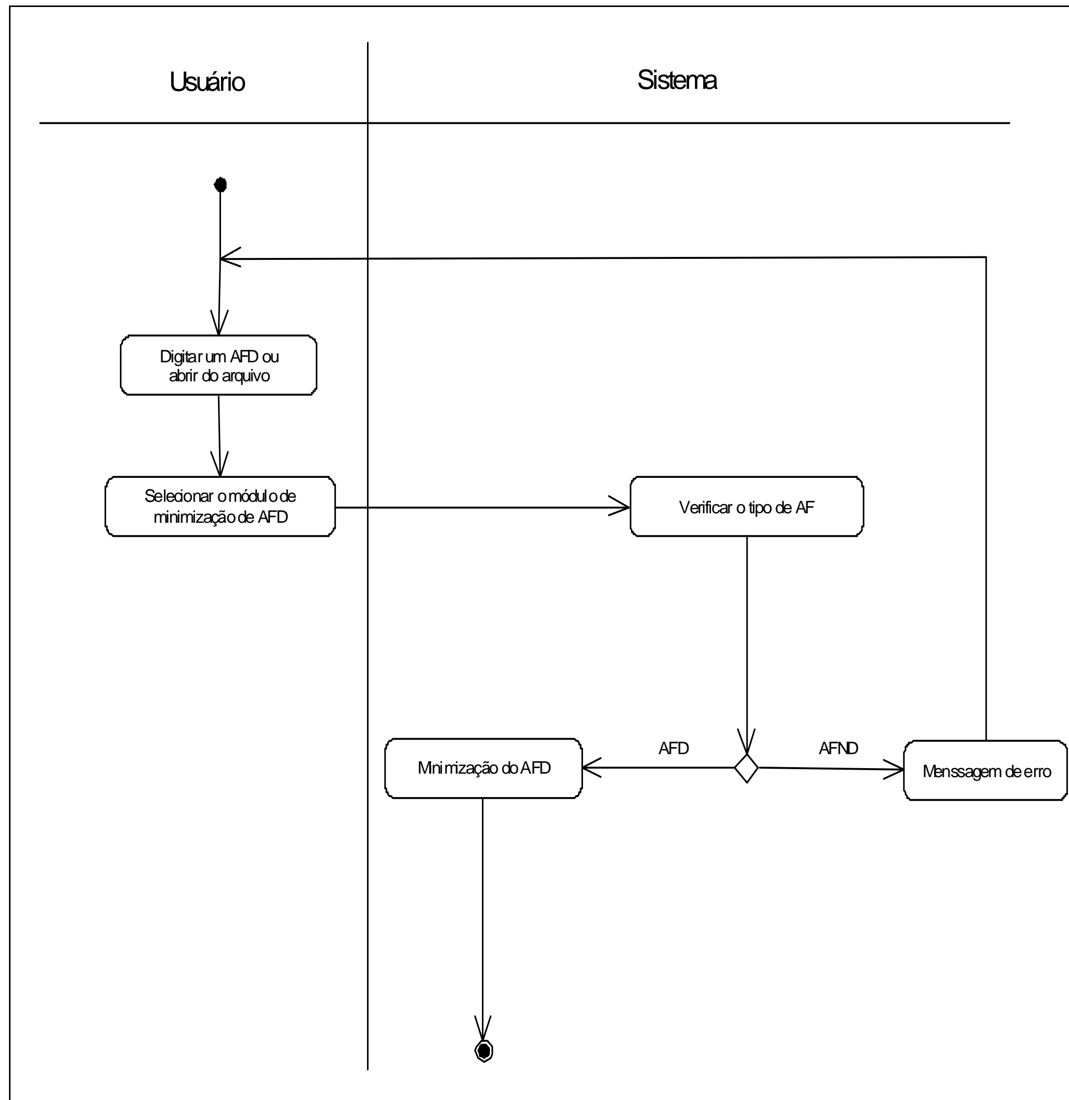


Figura 15. Diagrama de Atividades– Módulo de Minimização do AFD

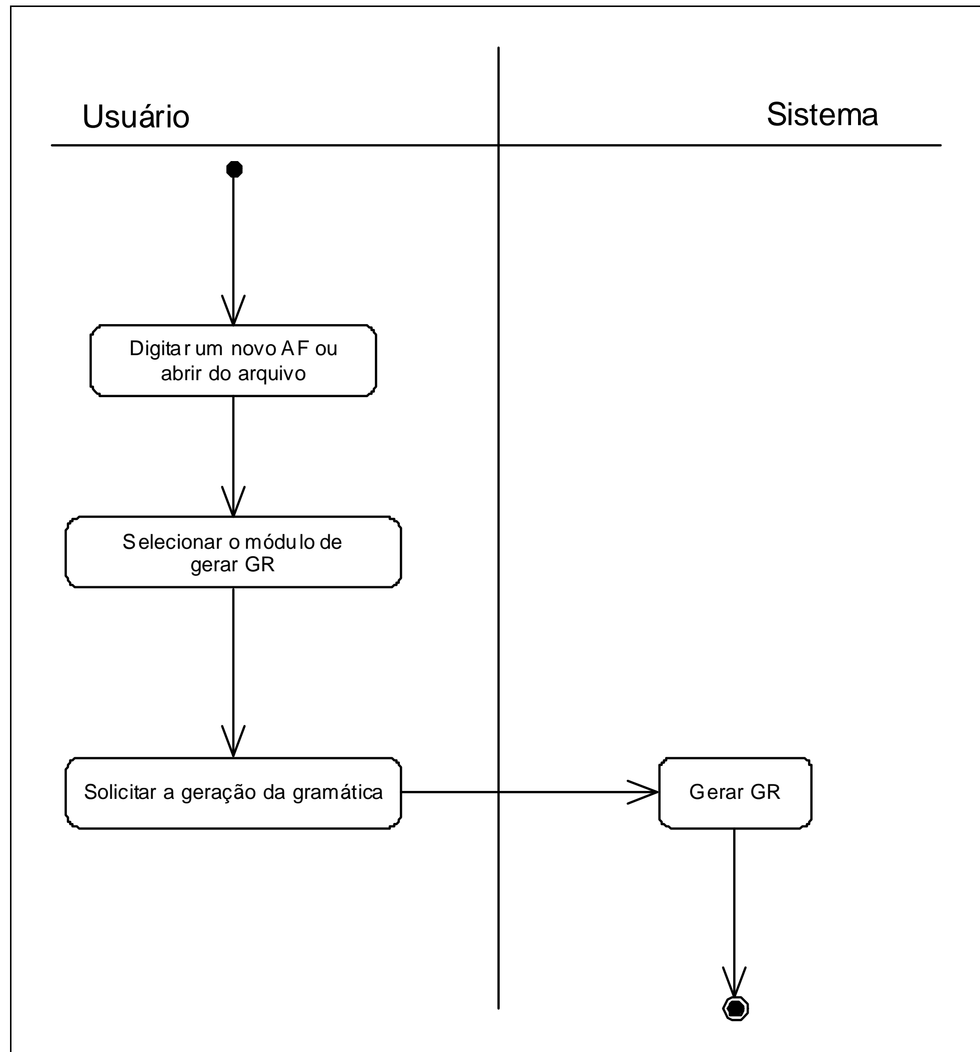


Figura 16. Diagrama de Atividades– Módulo de Geração de GR

Após as etapas acima descritas, ou seja, levantamento bibliográfico e modelagem em UML , realizou-se então a implementação dos módulos em linguagem *object pascal*, por meio da plataforma Borland® Delphi™ Enterprise, versão 7.0, sendo este utilizado devido ser um aperfeiçoamento da Shell AFLAB. Visto essa situação, foi necessário possuir o código fonte da ferramenta, versão 1.0.0. Quanto ao sistema operacional, foi utilizado o Microsoft® Windows XP.

4.3 DESENVOLVIMENTO

4.3.1 Módulo de Transformação de AFND em AFD

Para implementação da transformação do AFND em AFD, utilizou-se o procedimento já descrito nos capítulos anteriores, ou seja, foi colocado em prática o que já havia sido estudado, como segue:

a) determinar a primeira linha da tabela de transição do AFD como sendo o conjunto unitário contendo apenas o estado inicial do AFND;
b) verificar, para cada estado do AFD, se as transições foram determinadas, repetindo esse passo até que todas as transições tenham sido determinadas;
c) determinar o estado inicial do AFD como sendo a primeira linha da tabela de transições;
d) determinar o(s) estado(s) final(is) do AFD como sendo todos os estados (todas as linha) rotuladas com conjuntos que contenham pelo menos um estado final do AFND.

Para implementação das etapas acima descritas foram criadas duas classes para auxiliar no processo, são estas: *Testados_tranf* e *Tconexao_tranf*, que irão armazenar os estados e transições do AFD, respectivamente. A classe *Testados_tranf* armazena o nome do estado do AFD gerado na transformação e a identificação se o estado é inicial ou final. Já na classe *Tconexao_tranf* armazena o estado de origem e destino, que são os estados gerados na transformação, além de armazenar também o símbolo de entrada da transição. O código demonstrado na Figura 17 mostra o algoritmo onde essas classes e métodos são declarados.

```
type

    Testados_tranf = class
    nome_estado_tranf: string;
    estado_final: Boolean;
    Estado_inicial: Boolean;
    procedure NovoEstado_tranf(Rotulo: string; EFinais, Einicial: Boolean);
    function ContaEstados_tranf: integer;
    procedure ApagaEstados_tranf;
end;

    Tconexao_tranf = class
    estado_origem, estado_destino, Caractere :string;
    procedure NovaConexao_tranf(Origem, Destino, Rotulo: string);
    procedure ApagaConexoes_tranf;
    function ContaConexoes_tranf: integer;
    procedure ExcluiConexao_tranf(PosVet: integer);
end;
```

Figura 17. Estrutura de Armazenamento

Como já acontece no algoritmo da versão inicial do AFLAB, nesse módulo os vetores também são alocados dinamicamente, dessa forma não se tem um limite no número de estados do AF, como é demonstrado na declaração dos vetores na Figura 18.

```
var

  Estados_tranf: array of Testados_tranf;
  Conexao_tranf: array of Tconexao_tranf;
```

Figura 18. Tipos declarados

O algoritmo de transformação está contido na *procedure* nomeada de *transformação*. Iniciando esse algoritmo tem-se o primeiro passo do processo de transformação, que está contido na Figura 19. O algoritmo que determina o primeiro estado do AFD, no caso o estado inicial do AFND passa a ser inicial do AFD.

```
for i:=0 to ListaEstados.ContaEstados -1 do begin
      if (Estados[i].Estado_inicial=True) then begin
            ListaEstado2.NovoEstado_tranf(Estados[i].nome_estado,Estados[i].estado_final,Estados[i].Estado_inicial);
            for g:=0 to ListaConexoes.ContaConexoes -1 do begin
                  if (Conexao[g].estado_origem = Estados[i].nome_estado) then begin
                    ListaConexoes_tranf.NovaConexao_tranf(Conexao[g].estado_origem,
                    Conexao[g].estado_destino, Conexao[g].Caractere);
                  end;
            end;
      end;
end;
```

Figura 19. Definição do primeiro estado do AFD

A Figura 20 apresenta o algoritmo da segunda etapa do processo de transformação, verifica para cada estado do AFD, se as transições foram determinadas, ou seja, identificar se a união das transições de cada estado do AFND, com o mesmo símbolo de entrada, já está presente na tabela de estados do AFD, no caso armazenado no vetor *Testados_tranf*. Em caso negativo, é criado um novo estado do AFD, e para cada símbolo de entrada, é realizado a união das transições dos respectivos estados do AFND que compõe esse novo estado do AFD. Na Figura 21, é demonstrado a inclusão do estado oriundo da união das transições sendo inserido no AFD e sua identificação como final ou não. Lembrando que os estados do AFD são armazenados na classe *Testados_tranf* e as transições na classe *Tconexao_tranf*.

```
while (m<ListaEstados_tranf.ContaEstados_tranf+1)do begin
   for n:=0 to ListaEstado2.ContaEstados_tranf -1 do begin
       for j:=0 to ListaAlfabeto.ContaAlfabeto-1 do begin
         for p:=0 to  ListaConexoes2.ContaConexoes_tranf-1 do begin
             if (Conexao_tranf[p].estado_origem=Estados_tranf[n].nome_estado_tranf)
             and(Conexao_tranf[p].Caractere=Alfabeto[j].letra_alfabeto) then begin
                     status3[h]:=Conexao_tranf[p].estado_destino+',';
                         for x:=0 to ListaEstados.ContaEstados -1 do begin
                               if (Estados[x].nome_estado=Conexao_tranf[p].estado_destino) then begin
                                     if (Estados[x].estado_final=True) then begin
                                           status6[y]:=status6[y]+1;
                                     end;

                                     if (Estados[x].estado_final=False)or(Estados[x].Estado_inicial=True) then begin
                                           status6[y]:=status6[y]+0;
                                     end;
                               end;
                         end;
                     h:=h+1;
             end;
         end;
         ok2:=True;
         while (ok2<>false) do begin
                 ok2:=False;
                 indice2:=0;
                 while (indice2<h -1) do begin
                     if CompareStr(status3[indice2], status3[indice2 + 1]) > 0 then begin
                         valor4 := status3[indice2];
                         status3[indice2]:= status3[indice2+1];
                         status3[indice2+1]:= valor4;
                         ok2 := True;
                     end;
                     indice2:=indice2+1;
                 end;
         end;

         for i:=0 to h-1 do begin
                 if (status3[i]<>'') then begin
                     status2[a]:=status2[a]+status3[i];
                 end;
         end;


    a:=a+1;
    h:=0;
    y:=y+1;

 for h:=0 to a -1 do begin
      indicador:=0;
      for g:=0 to ListaEstados_tranf.ContaEstados_tranf-1 do begin
              if (status2[h]=Estados_tranf[g].nome_estado_tranf) or (status2[h]= '') then begin
                    indicador:=1;
              end;

              if (Estados_tranf[g].Estado_inicial=True)and (status2[h]=Estados_tranf[g].nome_estado_tranf+',') then begin
                 indicador:=1;
              end;
      end;
```

Figura 20. Identifica se a trasição já foi determinada

```
if (indicador=0) then begin
        if (status6[h]>0) then begin
            ListaEstado2.NovoEstado_tranf(status2[h],True,False);
        end;
        if (status6[h]=0) then begin
            ListaEstado2.NovoEstado_tranf(status2[h],False,False);
        end;
```

Figura 21. Inclusão do estado no AFD

Na Figura 22, está presente o algoritmo que identifica a transição de cada estado incluso no vetor *Testados_tranf* e essas transições são armazenadas no vetor *Tconexao_tranf*. A fim de evitar que houvesse duplicidade de estados no AFD após inserir a transição de cada estado é realizado uma ordenação.

```
        while i > 0 do begin
             S2:= Trim(Copy(S, 1, i - 1)); // Copia desde 1 até a virgula - 1
             Delete(S, 1, i); // Apaga desde 1 até a virgula
             if Length(S2) > 0 then  begin // Se copiou alguma coisa
                  for i:=0 to ListaConexoes.ContaConexoes-1 do begin
                      cont:=0;
                      if (Conexao[i].estado_origem=S2) then begin
                          for j:=0 to ListaConexoes_tranf.ContaConexoes_tranf -1 do begin
                                if ((Conexao_tranf[j].estado_destino)=(Conexao[i].estado_destino))
                                and(Conexao_tranf[j].estado_origem=status2[h])
                                and(Conexao_tranf[j].Caractere=Conexao[i].Caractere) then begin
                                    cont:=1;
                                end;
                          end;
                          if (cont=0) then begin
                              ListaConexoes2.NovaConexao_tranf(status2[h],Conexao[i].estado_destino,Conexao[i].Caractere);
                          end;
                      end;
                  end;
             end;
        i:= Pos(',', S); // Pega a posicao das proxima virgula

        end;
    ok:=True;
    while (ok<>false) do begin
        ok:=False;
        indice:=0;
   while (indice<ListaConexoes_tranf.ContaConexoes_tranf -1) do begin

         if CompareStr(Conexao_tranf[indice].estado_destino, Conexao_tranf[indice + 1].estado_destino) > 0 then
             begin
                   //ShowMessage('ordenar:');

                   valor1 := Conexao_tranf[indice].estado_origem;
                   valor2 := Conexao_tranf[indice].estado_destino;
                   valor3 := Conexao_tranf[indice].Caractere;
                   Conexao_tranf[indice].estado_origem:= Conexao_tranf[indice+1].estado_origem;
                   Conexao_tranf[indice].estado_destino:= Conexao_tranf[indice+1].estado_destino;
                   Conexao_tranf[indice].Caractere:= Conexao_tranf[indice+1].Caractere;
                   Conexao_tranf[indice+1].estado_origem:= valor1;
                   Conexao_tranf[indice+1].estado_destino:=valor2;
                   Conexao_tranf[indice+1].Caractere:=valor3;
                   ok := True;
               end;
     indice:=indice+1;
     end;
 end;
```

Figura 22. Inclusão das transições do estado no AFD

Após o término do processo de transformação o usuário pode solicitar que os estados sejam renomeados para facilitar a identificação. O *software* realiza isso automaticamente.

4.3.2 Módulo de Minimização de AFD

Como já visto no estudo realizado, o processo de minimização consiste na eliminação dos estados inacessíveis, mortos e equivalentes de um AFD. Baseado nessa premissa foi desenvolvido alguns algoritmos que realizam essas etapas.

Para eliminação dos estados inacessíveis desenvolveu-se a *procedure* chamado de *Elimina_estados_inacessiveis*, ilustrado na Figura 23, onde as condições para que o estado seja inacessível foi de o mesmo não ser inicial e não ser destino de nenhum estado. Após a identificação do estado inacessível é realizado a eliminação das transições desse estado por meio do método *RemoveConexoes* da classe *Tconexao* e o mesmo é inserido em um vetor (*ListaElimina*) com o método *NovoElimina* para posteriormente ser eliminado do AFD junto com outros estados que possam existir no AF.

```
procedure Tfrm_minimizacao.Elimina_estados_inacessiveis;

var
h, i, C, R, j, n, status, status1, status2 :integer;
S , T:string;
ListaElimina : TElimina;

begin
status:=0;
status:=0;
NElimina:=0;
for i:=0 to ListaEstados.ContaEstados-1 do
    begin
  for j := 0 to ListaConexoes.ContaConexoes -1    do
     begin
          if (Estados[i].nome_estado <> Conexao[j].estado_destino)and(Estados[i].estado_inicial=False) then begin
          status1:=status1+1;
          end;
          if (Estados[i].nome_estado = Conexao[j].estado_destino)and(Estados[i].nome_estado = Conexao[j].estado_origem)
          and(Estados[i].estado_inicial=False)then begin
          status1:=status1+1;
          end;
     end;
     if (status1=ListaConexoes.ContaConexoes) then begin
        ListaConexoes.RemoveConexoes(Estados[i].nome_estado);
        ListaElimina.NovoElimina(Estados[i].nome_estado);
     end;

   status1:=0;
end;
status:=0;
        for i:=0 to ListaElimina.ContaElimina -1 do
            begin
              for j := 0 to ListaEstados.ContaEstados -1 do
              begin
                if (Elimina[i].nome_estado=Estados[j].nome_estado) then begin
                    status:=j;
                end;
              end;
                ListaEstados.RemoveEstado(status);
            end;
 end;
```

Figura 23. Eliminação Estados Inacessíveis

Já para eliminação dos estados mortos desenvolveu-se a *procedure* chamada *Elimina_estados_mortos*, ilustrado na Figura 24, onde as condições para que o estado seja morto é de o mesmo não ser final ou inicial e não ser origem de nenhuma transição, só para ele mesmo. Após a identificação do estado morto é realizado a eliminação das transições desse estado por meio do método *RemoveConexoes* da classe *Tconexao* e o mesmo é inserido em um vetor (*ListaElimina*) com método *NovoElimina* para posteriormente ser eliminado do AFD junto com outros estados que possam existir no AF.

```
procedure Tfrm_minimizacao.Elimina_estados_mortos;

var
h, i, C, R, j, n, status, status1, status2 :integer;
S , T:string;
ListaElimina : TElimina;

begin
status1:=0;
status2:=0;

status:=0;
NElimina:=0;
for i:=0 to ListaEstados.ContaEstados-1 do
    begin
  for j := 0 to ListaConexoes.ContaConexoes -1    do
     begin
          if (Estados[i].nome_estado <> Conexao[j].estado_origem)and(Estados[i].estado_inicial=False)
          and (Estados[i].estado_final=False)then begin
            status1:=status1+1;
          end;

          if (Estados[i].nome_estado = Conexao[j].estado_destino)and(Estados[i].nome_estado = Conexao[j].estado_origem)
          and(Estados[i].estado_inicial=False)and (Estados[i].estado_final=False)then begin
            status1:=status1+1;
          end;
      end;

     if (status1=ListaConexoes.ContaConexoes)  then begin
         ListaConexoes.RemoveConexoes(Estados[i].nome_estado);
         ListaElimina.NovoElimina(Estados[i].nome_estado);
    end;
    status1:=0;
    status2:=0;
  end;
    status:=0;
         for i:=0 to ListaElimina.ContaElimina -1 do
             begin
               for j := 0 to ListaEstados.ContaEstados -1 do
               begin
                 if (Elimina[i].nome_estado=Estados[j].nome_estado) then begin
                     status:=j;
                      end;
                 end;
                   ListaEstados.RemoveEstado(status);
                 end;
```

Figura 24. Eliminação Estados Mortos

A terceira etapa do processo de minimização é a eliminação dos estados equivalentes do AFD, para realização desse processo foi desenvolvido a *procedure Classes_Equivalencia*.

Primeiramente é identificado os estados finais e não finais do AFD, e esses são armazenados em vetores da classe chamada *TClasse*. A Figura 25 mostra a declaração dessa classe e a Figura 26, mostra os estados sendo inseridos no vetor da

classe *TClasse* por meio do método *Insere_ClasseFinais*, este caso os estados finais e no método *Insere_ClasseNaoFinais* para os estados não finais.

```
TClasse = class
nome_estado1 :string;


procedure Insere_ClasseFinais(Rotulo1:string);
function ContaClasseFinais: integer;
procedure Insere_ClasseNaoFinais(Rotulo1:string);
function ContaClasseNaoFinais: integer;
end;
```

Figura 25. Declaração da *TClasse*

```
for i:=0 to ListaEstados.ContaEstados -1 do begin
      if (Estados[i].estado_final = True) then begin
        ListaClasseFinais.Insere_ClasseFinais(Estados[i].nome_estado);
      end;
      if (Estados[i].estado_final = False) then begin
        ListaClasseNaoFinais.Insere_ClasseNaoFinais(Estados[i].nome_estado);
      end;
end;
```

Figura 26. Inserção dos estados finais e não finais

Após a identificação dos estados finais e não finais é realizado um comparativo das transições com cada par de estados finais e não finais, a fim de identificar se a transição de uma dupla de estado, com o mesmo símbolo de entrada, se destina a um estado final ou não final. Verificando que as transições dos estados em questão se destinam a um estado final ou não final, essa dupla de estado é inserido em um vetor da classe *TClasse_equiv*, por meio do método *Insere_ClasseEquivalencia,* onde são inseridos o nome dos estados e a união desses estados, para posteriormente possivelmente se tornarem estados únicos. A Figura 27 demonstra algoritmo realizado com os estados finais e a Figura 28 com os estados não finais.

```
        for i:=0 to ListaClasseFinais.ContaClasseFinais-1  do begin
                for h:=i+1 to ListaClasseFinais.ContaClasseFinais-1  do begin
                  if(ClasseFinais[i].nome_estado1<>ClasseFinais[h].nome_estado1) then begin
                      for j:=0 to ListaConexoes.ContaConexoes-1 do begin
                          if (Conexao[j].estado_origem=ClasseFinais[i].nome_estado1) then begin
                                if (Conexao[j].estado_destino<>'') then begin
                                    for g:=0 to ListaEstados.ContaEstados -1 do begin
                                          if (Conexao[j].estado_destino = Estados[g].nome_estado) then begin
                                                if (Estados[g].estado_final=True) then begin
                                                      status1[e] := 1;
                                                end;
                                                if (Estados[g].estado_final=False)then begin
                                                      status1[e] := 2;
                                                end;
                                                e:=e+1;
                                          end;
                                    end;
                                end;
                          end;
                      end;

                      for k:=0 to ListaConexoes.ContaConexoes-1 do begin
                          if (Conexao[k].estado_origem=ClasseFinais[h].nome_estado1) then begin
                              if (Conexao[k].estado_destino<>'') then begin
                                  for g:=0 to ListaEstados.ContaEstados -1 do begin
                                     if (Conexao[k].estado_destino = Estados[g].nome_estado) then begin
                                          if (Estados[g].estado_final=True) then begin
                                                status2[d] := 1;
                                          end;
                                          if (Estados[g].estado_final=False)then begin
                                                status2[d] := 2;
                                          end;
                                          d:=d+1;
                                     end;
                              end;
                          end;
                    end;
                end;

                indicador:=0;
               if(d=e) then begin
                  for p:= 0 to  ListaAlfabeto.ContaAlfabeto -1 do begin
                      if (status1[p] = status2[p]) then begin
                          indicador:= indicador +1;
                      end;
                  end;
               end;
               if (indicador = ListaAlfabeto.ContaAlfabeto) then begin
                  ListaEquiv.Insere_ClasseEquivalencia(ClasseFinais[i].nome_estado1, ClasseFinais[h].nome_estado1,
                  ClasseFinais[i].nome_estado1+ClasseFinais[h].nome_estado1);
                  indice:=indice+1;
               end;
               d:=0;
               e:=0;

            end;
          end;
    indice:=0;
    end;
```

Figura 27. Identificação dos estados equivalentes finais

```
for i:=0 to ListaClasseNaoFinais.ContaClasseNaoFinais-1  do begin
        for h:=i+1 to ListaClasseNaoFinais.ContaClasseNaoFinais-1  do begin
          if(ClasseNaoFinais[i].nome_estado1<>ClasseNaoFinais[h].nome_estado1) then begin
              for j:=0 to ListaConexoes.ContaConexoes-1 do begin
                  if (Conexao[j].estado_origem=ClasseNaoFinais[i].nome_estado1) then begin
                      if (Conexao[j].estado_destino<>'') then begin
                          for g:=0 to ListaEstados.ContaEstados -1 do begin
                              if (Conexao[j].estado_destino = Estados[g].nome_estado) then begin
                                  if (Estados[g].estado_final=True) then begin
                                      status1[e] := 1;
                                  end;
                                  if (Estados[g].estado_final=False)then begin
                                      status1[e] := 2;
                                  end;
                                   e:=e+1;
                              end;
                          end;
                      end;
                  end;
              end;

              for k:=0 to ListaConexoes.ContaConexoes-1 do begin
                  if (Conexao[k].estado_origem=ClasseNaoFinais[h].nome_estado1) then begin
                    if (Conexao[k].estado_destino<>'') then begin
                        for g:=0 to ListaEstados.ContaEstados -1 do begin
                            if (Conexao[k].estado_destino = Estados[g].nome_estado) then begin
                                if (Estados[g].estado_final=True) then begin
                                    status2[d] := 1;
                                end;
                                if (Estados[g].estado_final=False)then begin
                                    status2[d] := 2;
                                end;
                                d:=d+1;
                            end;
                        end;
                    end;
                end;
            end;

        indicador:=0;
        if(d=e) then begin
            for p:= 0 to  ListaAlfabeto.ContaAlfabeto -1 do begin
                if (status1[p] = status2[p]) then begin
                    indicador:= indicador +1;
                end;
            end;
        end;
        if (indicador = ListaAlfabeto.ContaAlfabeto) then begin
            ListaEquiv.Insere_ClasseEquivalencia(ClasseNaoFinais[i].nome_estado1,
            indice:=indice+1;
        end;
        d:=0;
        e:=0;
    end;
  end;
indice:=0;
end;
```

Figura 28. Identificação dos estados equivalentes não finais

O próximo passo da eliminação dos estados equivalentes é verificar se a união das transições de cada par de estados, armazenados no vetor *TClasse_equiv*, se destinam com o mesmo símbolo de entrada, a estados que pertencem a mesma classe de equivalência. Caso isso seja negativo esse par de estados é eliminado do vetor *TClasse_equiv*, mas se positivo são unificadas suas transições e marcado como estado inicial ou final, dependendo do tipo dos estados. Na Figura 29, está o código dessa etapa do processo.

```
    while (b<ListaEquiv.ContaClasseEquivalencia) do begin
        for i:= 0 to ListaConexoes.ContaConexoes-1 do begin
            if (Conexao[i].estado_origem = Equivalencia[b].nome_estado1) then begin
                status3[n]:=Conexao[i].estado_destino;
                n:=n+1;
            end;
            if (Conexao[i].estado_origem = Equivalencia[b].nome_estado2) then begin
                status4[m]:=Conexao[i].estado_destino;
                m:=m+1;
            end;
        end;
        for i:=0 to ListaAlfabeto.ContaAlfabeto-1 do begin
            status7:=status3[i]+status4[i];
                if (status3[i]=status4[i]) then begin
                  status6:=status6+1;
                end;
                if (status3[i]<>status4[i]) then begin
                  for j:=0 to ListaEquiv.ContaClasseEquivalencia-1 do begin
                    if (status7=Equivalencia[j].estado_equiv) then begin
                        status6:=status6+1;
                    end;
                  end;
                end;
          end;
         if (status6<ListaAlfabeto.ContaAlfabeto) then begin
             ListaEquiv.Exclui_ClasseEquivalencia(b);
             b:=b-1;
         end;
    n:=0;
    m:=0;
    status6:=0;
    status7:='';
    b:=b+1;

end;
```

Figura 29. Identificação dos estados equivalentes

Após o término do processo de minimização o usuário pode solicitar que os estados sejam renomeados para facilitar a identificação. O *software* realiza isso automaticamente.

4.3.3 Módulo de Geração de Gramática Regular a partir de um Autômato Finito Determinístico

Nesse módulo o usuário tem a opção de geração da gramática a partir do autômato escolhido. O objetivo inicial deste trabalho era gerar gramática somente com AFD, mas o algoritmo desenvolvido permite também gerar a gramática de um AFND.

Para se gerar as regras gramaticais primeiramente percorrem-se os vetores das três classes principais do AFLAB, ou seja, *Testados*, *Talfabeto* e *Tconexao* a fim de identificar as transições de cada estado do AF. Cada transição é armazenada no vetor chamado *status3*. Após verificar todas transições possíveis do estado o vetor *status2* irá receber o somatório das informações armazenadas no *status3*, ou seja, o símbolo de entrada e o estado de destino de cada estado da classe *Testados*. A Figura 30 mostra o código desenvolvido para essa etapa.

```
for i:=0 to ListaEstados.ContaEstados-1 do begin
  for n:=0 to ListaAlfabeto.ContaAlfabeto -1 do begin
      for j:=0 to ListaConexoes.ContaConexoes-1 do begin

          if (Conexao[j].estado_origem=Estados[i].nome_estado)and(Conexao[j].Caractere=Alfabeto[n].letra_alfabeto) then begin

                if (h=0) then begin
                     for a:=0 to ListaEstados.ContaEstados -1 do begin
                         if (Estados[a].nome_estado=Conexao[j].estado_destino) then begin
                           if (Estados[a].estado_final=True) then begin
                               status3[h]:=Conexao[j].Caractere+Conexao[j].estado_destino+'/'+Conexao[j].Caractere;
                               indicador1:=1;
                           end;
                           if (Estados[a].estado_final=False) then begin
                               status3[h]:=Conexao[j].Caractere+Conexao[j].estado_destino;
                           end;
                         end;
                     end;
                end;

                if(h>0) then begin
                      for a:=0 to ListaEstados.ContaEstados -1 do begin
                          if (Estados[a].nome_estado=Conexao[j].estado_destino) then begin
                            if (Estados[a].estado_final=True)then begin
                                if (indicador1=0) then begin
                                status3[h]:='/'+Conexao[j].Caractere+Conexao[j].estado_destino+'/'+Conexao[j].Caractere;
                                indicador1:=1;
                                end
                                else begin
                                status3[h]:='/'+Conexao[j].Caractere+Conexao[j].estado_destino;
                                end;
                            end;
                            if (Estados[a].estado_final=false) then begin
                                status3[h]:='/'+Conexao[j].Caractere+Conexao[j].estado_destino;
                            end;
                          end;
                      end;
                end;
          h:=h+1;
          indicador:=1;
          end;
      end;
  indicador1:=0;
  end;
      if (indicador=0) then begin
          status2[p]:='î';
      end;

      for m:=0 to h-1 do begin
        status2[p]:=status2[p]+status3[m];
      end;
```

Figura 30. Gera gramática a partir de um AF

4.4 RESULTADOS OBTIDOS

O usuário digita o AF na Forma Tabular ou Gráfica se o mesmo for um AFD o usuário poderá minimizá-lo e gerar a gramática desse AF e se caso ele for um AFND o usurário poderá transformá-lo em determinístico, gerar a gramática desse autômato e

minimizá-lo após a transformação. Todas essas opções são oferecidas para usuário por meio de abas na tela principal do AFLAB.

A Figura 31 mostra a tela de minimização, onde o usuário pode clicar na aba *Minimização AF* e o sistema gera automaticamente o AFD minimizado e esse é visualizado pelo usuário em um *String Grid*. Após a minimização, o usuário pode solicitar que os estados sejam renomeados, bastando o mesmo clicar no botão *Renomear Estados* e o *software* fará isso automaticamente, como ilustra a Figura 32.

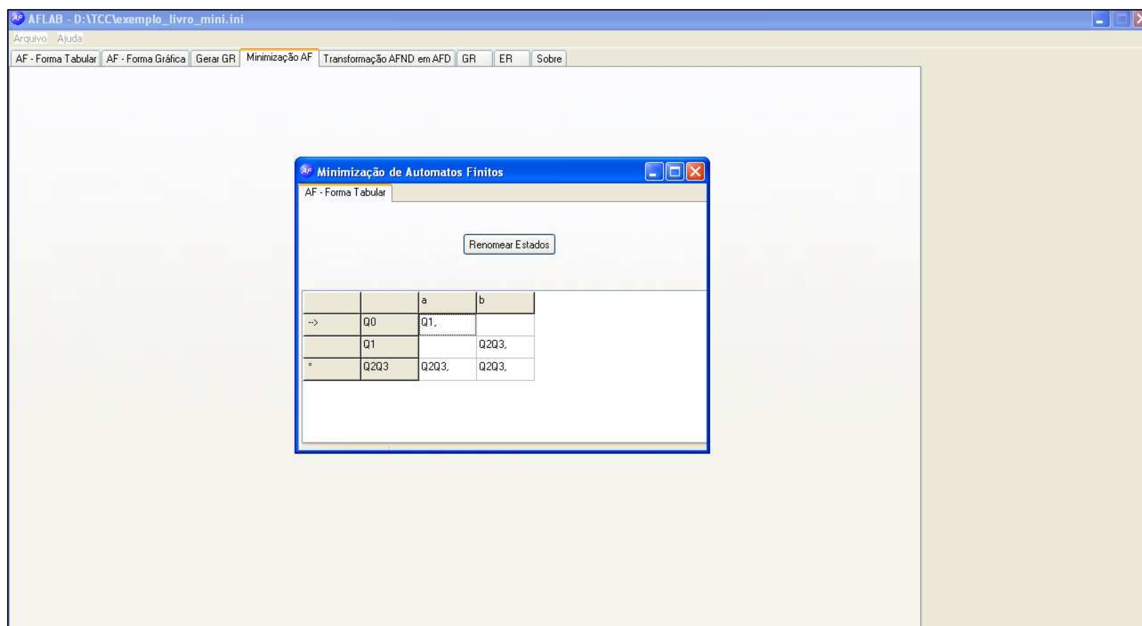


Figura 31. Aba minimização AFD

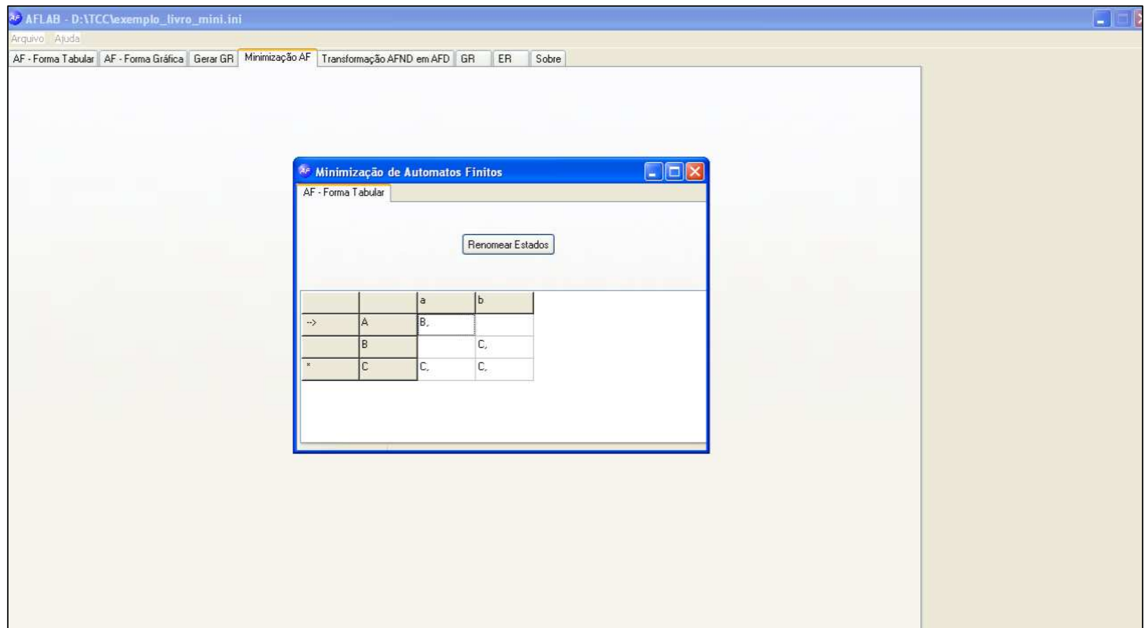


Figura 32. Aba minimização AFD renomeando os estados

A Figura 33 ilustra a tela de geração de GR, quando o usuário clicar na aba *Gerar GR* aparecerá um botão identificado como *Gerar Gramática*, onde o usuário clicando nesse botão o sistema gerará a gramática do AF digitado, que pode ser determinístico ou não determinístico.

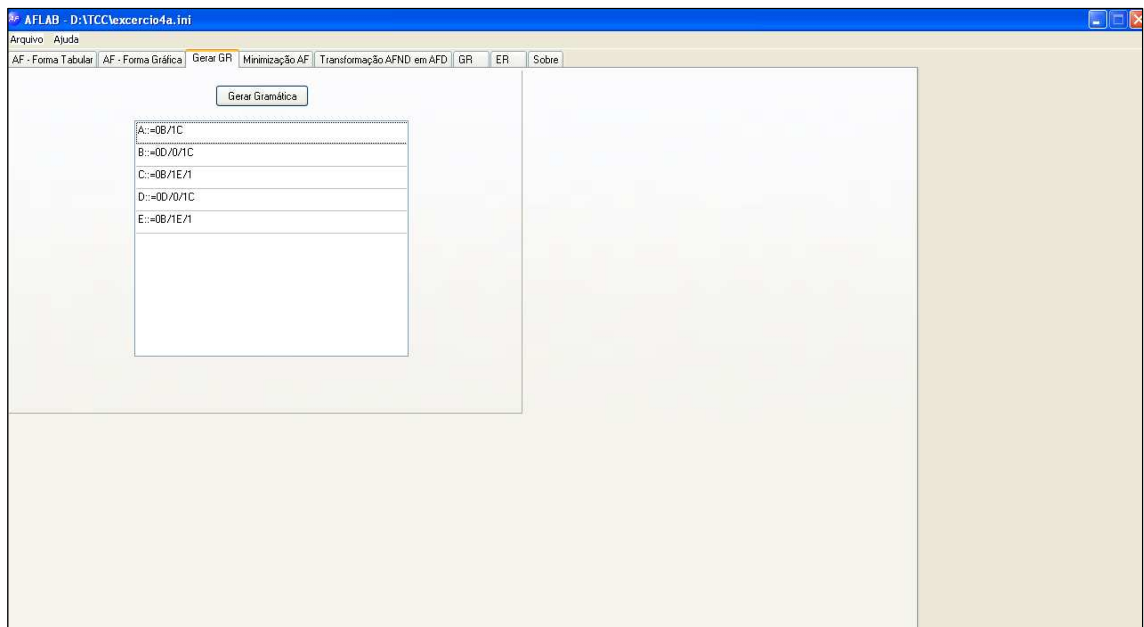


Figura 33. Tela de Geração de GR

Já a Figura 34 mostra a tela de transformação do AFND em AFD. O usuário quando clicar na aba *Transformação AFND em AFD*, o sistema irá gerar

automaticamente o AFD e este é visualizado pelo usuário em um *String Grid*. Após a transformação, o usuário pode solicitar que os estados sejam renomeados, bastando o mesmo clicar no botão *Renomear Estados* e o *software* fará isso automaticamente, como ilustra a Figura 35.

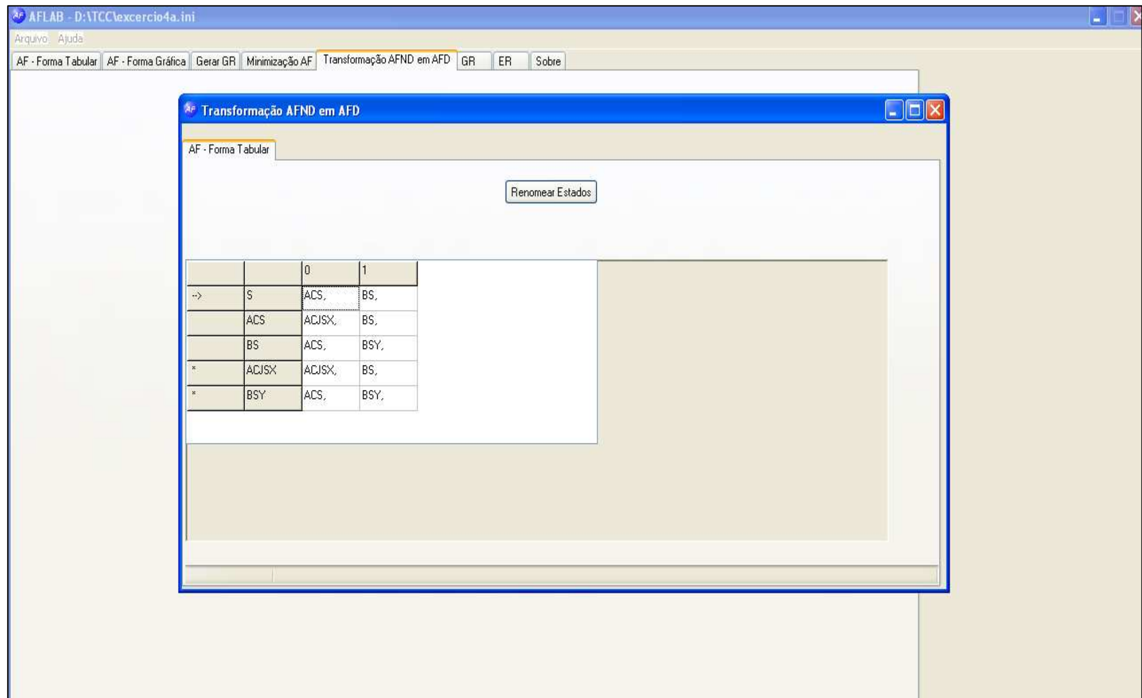


Figura 34. Tela de transformação de AFND em AFD

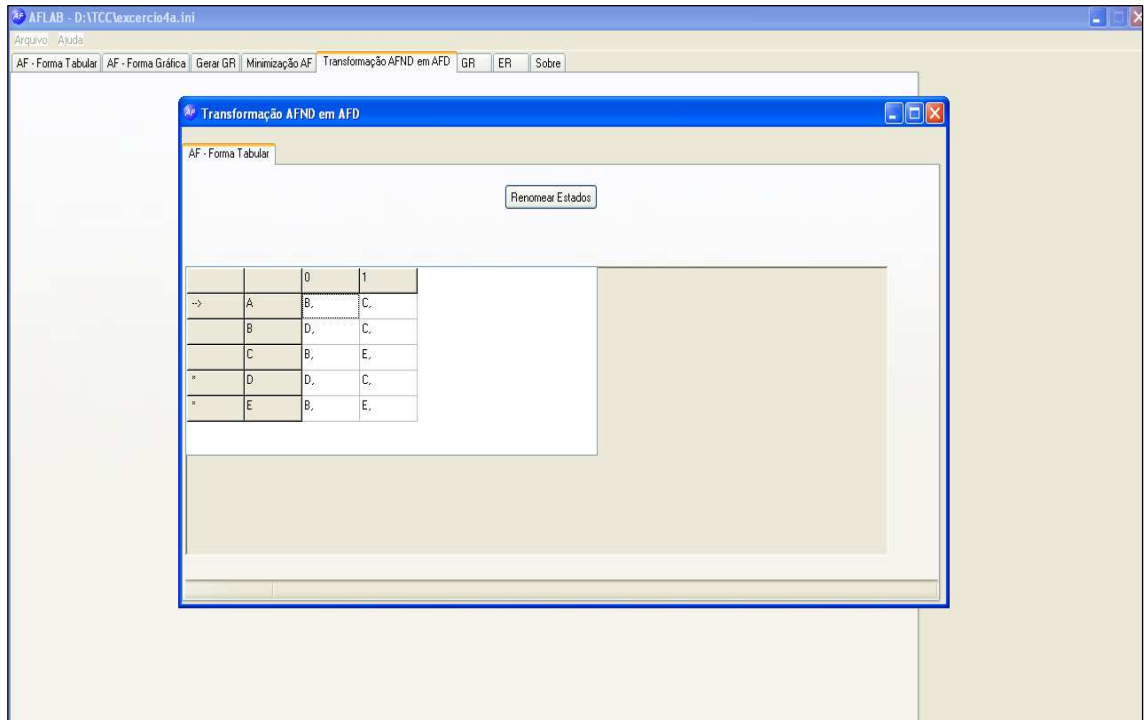


Figura 35. Tela de transformação do AFND em AFD com estados renomeados

Durante e após a implementação, foram realizados diversos testes em todos os módulos para verificar o funcionamento e a possível existência de erros. O funcionamento do *software* ficou como o esperado e erros não foram encontrados. Os módulos desenvolvidos atingiram o propósito apresentado para este trabalho, cumprindo o objetivo geral e os objetivos específicos.

CONCLUSÃO

As disciplinas de Linguagens Formais possuem uma escassez de ferramentas de apoio ao aprendizado desse tema. O desenvolvimento de softwares educacionais possibilita, sobretudo ao aluno a motivação para o estudo da disciplina, permitindo testar rapidamente novas idéias ou conceitos já ensinados, além disso, oferece a correlação do conteúdo teórico ao prático. Partindo dessa idéia que o AFLAB vem cada vez mais sendo aperfeiçoado com novas funcionalidades.

O AFLAB teve seu aperfeiçoamento neste trabalho permitindo agora ao usuário a transformação do AFND em AFD, minimização do AFD e geração da gramática a partir de um AFD, sendo esses os objetivos específicos alcançados com este trabalho. Vale ressaltar que a geração de gramática também é permitida para um AFND o que tornou ainda mais completo esse módulo.

Para não comprometer a implementação dos próximos módulos que possam surgir, não foram modificadas as classes originadas da primeira versão do AFLAB. Foram apenas criadas novas classes para os módulos de transformação do AFND em AFD e minimização do AFD.

Os módulos de transformação do AFND em AFD e geração de gramática a partir de um AF foram implementados com maior facilidade, pois havia de forma clara na fundamentação teórica os passos necessários para estes processos. Já na implementação do módulo de minimização do AFD, os passos realização deste processo se tornaram um tanto confusos no momento da implementação. Para suprir essa dificuldade foi desenvolvida uma lógica própria para o módulo de minimização do AFD.

Finalizando sugere-se para trabalhos futuros que o código seja transformado em linguagem multiplataforma possibilitando a execução do software em outros sistemas operacionais. Outra sugestão seria divulgar por meio eletrônico o projeto AFLAB para que outras universidades possam utilizar o software e desenvolver novas ferramentas de aprendizado.

REFERÊNCIAS

BOVET, Jean**. VAS – Visual Automata Simulator.** University of San Francisco, USFCA, Estados Unidos, 2004. Disponível em: http://www.cs.usfca.edu/%7Ejbovet/vas.html, Acessado em 20 de maio de 2009.

CANTÚ, Marco. **Dominando o Delphi 6**: a bíblia. Tradução de João Eduardo Nóbreg Tortello. São Paulo: Makron Books, 2002.

FOWLER, Martin; SCOTT, Kendal. **UML essencial**: um breve guia para a linguagem padrão de modelagem de objetos. Porto Alegre: Bookman, 2000. Tradução de Vera Pezerico e Christian Thomas Price.

FURTADO, Olinto José Varela. **Linguagens formais e compiladores**. Apostila da disciplina de Linguagens Formais e Compiladores da UFSC, 2004.

GAIDZINSKI, Marco Aurélio. **Ambiente de criação e manipulação de autômatos finitos na forma gráfica ou tabular para o reconhecimento de sentenças**. 2007. 61 f. Trabalho de Conclusão de Curso (Bacharelado) – Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, 2007.

GERSTING, Judith L. **Fundamentos matemáticos para a ciência da computação**. 3.ed. Rio de Janeiro: LTC, 1995.

HOPCROFT, John E.; ULLMAN, Jeffrey D.; MOTWANI, Rajeev. **Introdução à teoria de autômatos, linguagens e computação**. Tradução de Vandenberg D. de Souza. Rio de Janeiro: Elsevier, 2002.

JUKEMURA, Anibal S.; NASCIMENTO, Hugo A. D.; Uchoa, Joaquim Q. **GAM - Um Simulador para Auxiliar o Ensino de Linguagens Formais e de Autômatos**. In: XXV Congresso da Sociedade Brasileira de Computação, 2005, São Leopoldo, RS. Disponível em: <http://www.unisinos.br/_diversos/congresso/sbc2005/_dados/anais/pdf/arq0043.pdf>. Acesso em 20 maio de 2009.

LOUDEN Kenneth C.; **Compiladores:** princípios e práticas. Tradução de Flávio Soares Correia da Silva. São Paulo: Pioneira Thomson Learning, 2004.

MENEZES, Paulo Fernando Blauth. **Linguagens formais e autômatos**. 5.ed. Porto Alegre: Sagra Luzzatto, 2005.

RODGER, Susan H. **Java Formal Language and Automata Package (JFLAP)**. Rensselaer Polytechnic Institute, RPI, Estados Unidos, 1990. Disponível em: <http://www.jflap.org>. Acesso em: 19 set. 2007.

SANTOS, Fernando dos; VARGAS, Karly Schubert. **Ferramenta para a Transformação de Autômato Finito Não-Determinístico em Autômato Finito Determinístico.** In: XIII SEMINCO – Seminário de Computação, 2004, Blumenau, SC. Disponível em: <http://www.inf.furb.br/seminco/2004/artigos/134-vf.pdf>. Acesso em 01 jun de 2009.

SCARPATO, Christine Vieira. **Linguagens Formais e Compiladores**. Apostila da disciplina de Linguagens Formais e Compiladores da UNESC, 2004.

VIEIRA Newton José. **Introdução aos Fundamentos da Computação: Linguagens e Máquinas**. São Paulo: Pioneira Thomson Learning, 2006.

VIEIRA, Luiz Filipe Menezes; VIEIRA, Marcos Augusto Menezes; VIEIRA, Newton José. **Language Emulator, uma ferramenta de auxílio no ensino de Teoria da Computação**. In: II Workshop de Educação em Computação e Informática do Estado de Minas Gerais, 2003, Poços de Caldas, MG. Disponível em: <http://www.inf.pucpcaldas.br/eventos/weimig2003/ArtigosWEIMIG2003/WEIMIG2003LuizFilipeMenezes.pdf>. Acesso em: 10 jun. 2009.